UNIT IV

CHAPTER 4

Naïve Bayes and Support Vector Machine

Syllabus Topics

Bayes' Theorem, Naïve Bayes' Classifiers, Naïve Bayes in Scikit- learn - Bernoulli Naïve Bayes, Multinomial Naïve Bayes, and Gaussian Naïve Bayes.

Support Vector Machine (SVM) : Linear Support Vector Machines, Scikit - learn implementation- Linear Classification, Kernel based classification, Non-linear Examples. Controlled Support Vector Machines, Support Vector Regression.

4.1 Introduction to Naïve Bayes

- Naïve Bayes is a classification algorithm based on Bayes Theorem and the Maximum A Posteriori (MAP) hypothesis.
- It is useful for large data sets.
- It is also called as Idiot Bayes and Simple Bayes.
- Naïve Bayes makes an assumption that the effect of an attribute value on a given class is independent of the values of the other attributes. This assumption is know as class conditional independence.
- The above assumption is made to simplify the computation and in this sense it's called as Naïve(it means the data attributes are independent).

Syllabus Topic : Bayes' Theorem

4.1.1 Bayes Theorem

- Consider $X=\{x_1,x_2,x_3...x_n\}$ as a sample, whose components represent values made on a set of n attributes.
- In Bayes Theorem X is considered evidence. Let H be a Hypothesis, such that data X belongs to a class C.
- In classification, the goal is to determine P(H|X), the probability that the hypothesis H holds given the evidence (observed data sample X) .
- In other words, the probability that sample X belongs to class C.
- P(H|X) is called as a Posteriori probability of H conditioned on X, and is given as :

$$P(H|X) = \frac{P(X|H)\,P(H)}{P(X)}$$

Where P(H) is the a priori Probability which is independent of X.

P(X) is the a priori probability of X.

P(X|H) is the a posteriori probability of X conditioned on H.

- Let us understand this concept with the help of an example.
- Suppose we have a data set of customers. In this dataset age and income are the attributes and buys_computer as a class with values as Yes and No.
- Let us consider a sample X{ age : 45 years, income : Rs. 50,000}, we have to classify this unseen tuple as buys_computer = yes or buys_computer = No.
- P(H|X) is the probability that the customer X will buy a computer given his age and income, in our case age = 45 and income = Rs. 50,000.
- P(H) is the probability any customer will buy a computer regardless of his age and income, it is independent of X.
- P(X) is the probability from the data set that the customer is 45 years old and earns Rs. 50,000.
- P(X|H) is the probability that a customer , is 45 years old and he earns Rs. 50,000, given that he will buy a computer.

4.1.2 Maximum A Posteriori (MAP) Hypothesis

Based on Bayes Rule, we can compute the Maximum a Posteriori Hypothesis for the data:

$$h_{map} = \arg\max_{h \in H} P(h|D)$$

$$= \arg\max_{h \in H} \frac{P(D|h)\,P(h)}{P(D)} = \arg\max_{h \in H} P(D|h)\,P(h)$$

H : set of all Hypothesis

P(D) can be dropped as the probability of the data is constant(and independent of hypothesis)

4.1.3 Maximum Likelihood

- Now assume that all hypotheses are equally probable a priori, i.e., $P(h_i) = P(h_j)$ for all $h_i, h_j \in H$.
- This is called assuming a uniform prior. It simplifies computing the posterior.

$$h_{ML} = \arg\max_{h \in H} P(D|h)$$

- This hypothesis is called maximum likelihood hypothesis.

Syllabus Topic : Naïve Bayes' Classifiers

4.2 Naïve Bayes Classifier

Following is the working of Naïve Bayesian Classifier :

1. Let D be a training set of tuples along with their class labels. Each tuple is represented by an n-dimensional vector, $X = (x_1, x_2, x_3,....x_n)$, consisting of n measurements on the tuple from n attributes $A_1, A_2,....A_n$.
2. Consider there are m classes, $C_1, C_2,....C_m$. Given a tuple X the classifier will predict that X belongs to a class having highest posterior probability conditioned on X which is given as

$$P(C_i|X) > P(C_j|X) \quad \text{for } 1 \cdot j \cdot m, j \neq i$$

Hence we maximize $P(C_i|X)$. The class C_i for which $P(C_i|X)$ is maximized is called the maximum posteriori hypothesis. By Bayes theorem we have

$$P(C_i|X) = \frac{P(X|C_i)\,P(C_i)}{P(X)}$$

3. As $P(X)$ is constant for all classes, only the $P(X|C_i)\,P(C_i)$ needs to be maximized. In case the prior probabilities of class are not known then it is likely that the probabilities are equal i.e. $P(C_1) = P(C_2) = P(C_3) = \ldots = P(C_m)$ and therefore only $P(X|C_i)$ needs to be maximized else $P(X|C_i)\,P(C_i)$ needs to be maximized. The class prior probabilities can be calculated as

$$P(C_i) = |C_{i,D}| / |D|$$

Where $|C_{i,D}|$ is the number of training tuples of class C_i in D.

4. It is computationally expensive to compute $P(X|C_i)$, given a data set with many attributes. To reduce the computational complexity, naïve assumption of class conditional independence is made (No dependence relationships among attributes), thus

$$P(X|C_i) = \prod_{k=1}^{n} P(x_k|C_i)$$

$$= P(x_1|C_i) \times P(x_2|C_i) \times \ldots\ldots \times P(x_n|C_i)$$

These probabilities can be estimated $P(x_1|C_i)$, $P(x_2|C_i)$, $P(x_3|C_i)$,...$P(x_n|C_i)$ from the training tuples. x_k is the value of attribute A_k for tuple X.

Computation of $P(X|C_i)$ is dependent on whether the attribute is categorical or continuous – valued.

(a) If A_k is Categorical : $P(x_k|C_i)$ is the number of tuples of class C_i in D having the value x_k for A_k, divided by $|C_{i,D}|$, the number of tuples of class C_i in D.

(b) If A_k is Continuous : A continuous valued attribute has a Gaussian distribution with a mean μ and standard deviation σ, defined by

$$g(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}};$$

$$P(x_k|C_i) = g(x_k, \mu C_i, \sigma C_i)$$

Compute μC_i and σC_i, which are the mean (i.e., average) and standard deviation, respectively, of the values of attribute A_k for training tuples of class C_i. Use these values in above equation to calculate $P(x_k|C_i)$.

5. For Prediction of class label of X ,$P(X|C_i)P(C_i)$ is evaluated for each class C_i. The classifier predicts that the class label of tuple X is the class C_i if and only if

$$P(X|C_i)P(C_i) > P(X|C_j)P(C_j) \text{ for } 1 \le j \le m, j \ne i.$$

Which means the predicted class label is the class C_i for which $P(X|C_i)P(C_i)$ is the maximum.

4.2.1 E.g. Naïve Bayesian Classifier

Predict a class label of an unknown sample using Naive Bayesian classification on the following training dataset from all electronics customer database.

Given a training set, we can compute the probabilities as follows :

- The classification problem may be formalized using a-posterior probabilities :

  P(C|Y) is the probability that the sample tuple $Y = \langle y_1, \ldots, y_k \rangle$ is of class C.

- Assign to sample Y the class label C such that P(C|Y) is maximal.

Age	Income	Student	Credit_rating	Class: buys_computer
<=30	High	No	Fair	No
<=30	High	No	Excellent	No
31...40	High	No	Fair	Yes
>40	Medium	No	Fair	Yes
>40	Low	Yes	Fair	Yes
>40	Low	Yes	Excellent	No
31...40	Low	Yes	Excellent	Yes
<=30	Medium	No	Fair	No
<=30	Low	Yes	Fair	Yes
>40	Medium	Yes	Fair	Yes
<=30	Medium	Yes	Excellent	Yes
31...40	Medium	No	Excellent	Yes
31...40	High	Yes	Fair	Yes
>40	Medium	No	Excellent	No

The unknown sample is x={ age= "<=30" , Income= "Medium", Student= "Yes", Credit rating= "Fair"}

Age	
P(<=30|yes) = 2/9	P(<=30|No) = 3/5
P(31...40|yes) = 4/9	P(31...40| No) = 0
P(>40|yes) = 3/9	P(>40| No) = 2/5
Income	
P(High|yes) = 2/9	P(High|No) = 2/5
P(Medium|yes) = 4/9	P(Medium| No) = 2/5
P(low|yes) = 3/9	P(low| No) = 1/5

Student	
P(Noyes) = 3/9	P(NoNo) = 4/5
P(yesyes) = 6/9	P(yesNo) = 1/5
Credit_Rating	
P(fairyes) = 6/9	P(fairNo) = 2/5
P(excellentyes) =3/9	P(excellentNo) = 3/5

P(yes) = 9/14
P(No) = 5/14

An unseen sample X = < age= "<=30",Income= "Medium", Student= "Yes", Credit rating= "Fair" >

$$P(X|Yes) \cdot P(Yes) = P(Age<="30"| Yes) \cdot P(Income=Medium| Yes)$$

$$\cdot P(Student="yes"| Yes) \cdot P(Credit\ Rating="fair"| Yes)$$

$$\cdot P(yes) = 2/9 \cdot 4/9 \cdot 6/9\ 6/9.9/14 = 0.028$$

$$P(X|No) \cdot P(No) = P(Age<="30"| No) \cdot P(Income=Medium| No)$$

$$P(Student="yes"| No) \cdot P(Credit\ Rating="fair"| No)$$

$$\cdot P(No) = 0.007$$

Since 0.028 > 0.007, Therefore the naive Bayesian classifier predicts Buys computer = "Yes" for sample X

4.2.2 Properties of Bayes Classifiers

1. **Instrumentality**
   - The prior and likelihood can be updated dynamically with each training example.
   - Flexible and robust to errors.
2. **Combines prior knowledge and observed data**
   - Given the training data, the prior probability of a hypothesis is multiplied with probability of the hypothesis.
3. **Probabilistic hypotheses**
   - The output of Bayes classifier includes classification as well as probability distribution over all classes.
4. **Meta-classification**
   - The output of several classifiers can be combined together.
   - For e.g. the probabilities of all classifiers predicted for a given class can be multiplied together.

Syllabus Topic : Naïve Bayes in Scikit-Learn

4.3 Bayes in Scikit-Learn

- Based on the same number of different probabilistic distributions, scikit – learn provides implementation of three variants of Naïve bayes : Bernoulli Naïve Bayes, Multinomial Naïve bayes and Gaussian Naïve Bayes.
- Bernoulli naïve bayes is a binary distribution and this type is useful when a feature is present or absent.
- Multinomial Naïve Bayes is a discrete distribution and is useful when a feature needs to represented using a whole number.
- The Gaussian distribution is a continuous distribution and is characterised by mean and variance.

Syllabus Topic : Bernoulli Naïve Bayes

4.4 Bernoulli Naïve Bayes

- Consider X is a random variable having Bernoulli distribution.
- It can assume only two values (say for e.g. 0 and 1) and their probability is given as follows :

$$P(X) = \begin{cases} p & \text{if } X = 1 \\ q & \text{if } X = 0 \end{cases}$$

  Where $q = 1 - p$ and $0 < p < 1$

- E.g. : Implementation of Bernoulli Naïve Bayes in Scikit-Learn
- Generating a dummy data set :

```
from sklearn.datasets import make_classification
nb_samples = 300
X, Y = make_classification(n_samples=nb_samples, n_features=2, n_informative=2, n_redundant=0)
```

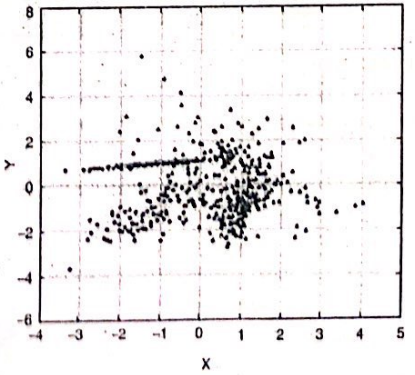


Fig. 4.4.1 : Dummy data generated

- Bernoulli naive Bayes needs binary feature vectors.
- The binarize parameter in BernoulliNB class allows using a threshold that can be used internally to transform the features to binary
- Let us use 0.0 as the binary threshold.

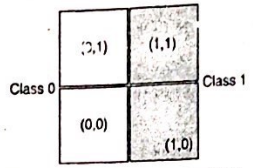


Fig. 4.4.2 : Internally binarized data

```
from sklearn.naive_bayes import BernoulliNB
from sklearn.model_selection
import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25)
bnb = BernoulliNB(binarize=0.0)
bnb.fit(X_train, Y_train)
print(bnb.score(X_test, Y_test))
```

Output

```
0.8,333333333333339
```

Checking for Prediction

```
data = np.array([[3, 0], [0, 1], [1, 0], [1, 1]])
print(bnb.predict(data))
```

Output

```
array([0, 0, 1, 1])
```

Syllabus Topic : Multinomial Naïve Bayes

4.5 Multinomial Naïve Bayes

- Multinomial Naïve Bayes is a model used when the value represents the number of occurrences of a term or its relative frequency.
- If a feature vector has n elements and each of them have k different values with probability p_i then

$$P(X_1 = x_1 \cap X_2 = x_2 \cap .... \cap X_k = x_k) = \frac{n!}{\prod_i x_i!} \prod_i p_i^{x_i}$$

- To prevent the model from setting null probabilities when the frequency is zero , an alpha paremeter (called Laplace smoothing factor) is used whose default value is 1.0
- E.g in scikit-learn. There are two records a first record is of city and the second one of countryside.
- The output class for city is 1 and countryside is 0.


```
from sklearn.feature_extraction import DictVectorizer
data = [ {'house': 100, 'street': 50, 'shop': 25, 'car': 100, 'tree': 20}, {'house': 5, 'street': 5, 'shop': 0, 'car': 10, 'tree': 500, 'river':
1} ]
dv = DictVectorizer(sparse=False)
X = dv.fit_transform(data)
Y = np.array([1, 0])
>>>X
#OUTPUT
array([[ 100.,  100.,   0.,  25.,  50.,  20.], [  10.,   5.,   1.,   0.,   5., 500.]])
```

- The term river is missing from the first set, so alpha is set to 1.0.
- Training the Multinomial Naïve Bayes

```
from sklearn.naive_bayes import MultinomialNB
mnb = MultinomialNB()
mnb.fit(X, Y) MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

Testing the model

```
test_data = data = [ {'house': 80, 'street': 20, 'shop': 15, 'car': 70, 'tree': 10, 'river': 1},
{'house': 10, 'street': 5, 'shop': 1, 'car': 8, 'tree': 300, 'river': 0} ]
mnb.predict(dv.fit_transform(test_data))
#Output
array([1, 0])
```

Syllabus Topic : Gaussian Naïve Bayes

4.6 Gaussian Naïve Bayes

- Gaussian Naïve Bayes is used when the values are continuous whose probabilities can be modeled using a Gaussian Distribution.
- Conditional Probabilities are also Gaussian Distributed, mean and variance needs to be estimated of each of them using maximum likelihood approach.
- Using the Gaussian property we get,

$$L(\mu; \sigma^2; x_i \mid y) = \log \prod_k P(x_i^{(k)} \mid y) = \sum_k \log P(x_i^{(k)} \mid y)$$

k- refers to the sample in the data set

- **An example, comparing** Gaussian Naive Bayes with logistic regression using the ROC curves. The dataset has 300 samples with two features. Each sample belongs to a single class :

```
from sklearn.datasets import make_classification
nb_samples = 300
X, Y = make_classification(n_samples= nb_samples, n_features=2, n_informative=2, n_redundant=0)
```

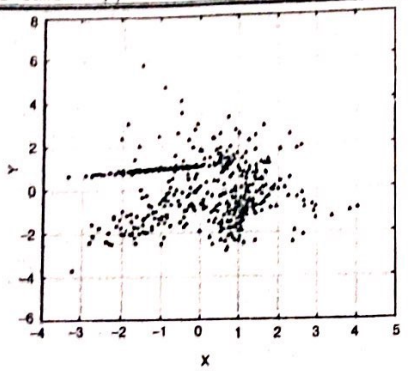


Fig. 4.6.1 : A plot of dataset

Training both the models and generating ROC curves

```
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25) >>>gnb = GaussianNB()
gnb.fit(X_train, Y_train)
Y_gnb_score = gnb.predict_proba(X_test)
lr = LogisticRegression()
lr.fit(X_train, Y_train)
Y_lr_score = lr.decision_function(X_test)
fpr_gnb, tpr_gnb, thresholds_gnb = roc_curve(Y_test, Y_gnb_score[:, 1])
fpr_lr,tpr_lr, thresholds_lr = roc_curve(Y_test, Y_lr_score)
```

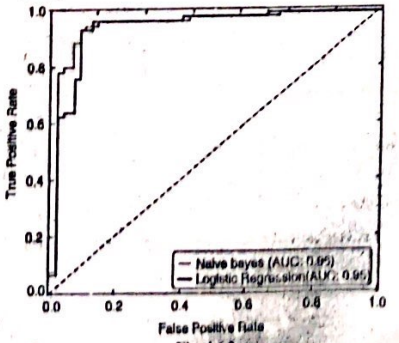


Fig. 4.6.2

As can be seen from the Fig. 4.6.2, Naïve bayes performs slightly better than Logistic regression, both the classifiers have similar accuracy and AUC (Area Under Curve).

4.7 Application of Naïve Bayes

1. **Real-time Prediction**

   Due to high speed of Naïve bayes algorithm, it can be used for making predictions in real time.

2. **Multi-class Prediction**

   The posterior probability of multiple classes of the target variable can be predicted by the algorithm.

3. **Text classification/ Spam Filtering/ Sentiment Analysis**

   - Due to their better results in multiclass problems and independence rule, Naïve Bayes classifier is suitable for text classification.
   - It has a high success rate compared to other algorithms.
   - Due to its success rates they are widely used in spam filtering and sentiment analysis applications.

4. **Recommendation System**

   - Naïve Bayes can be combined together with collaborative filtering to make a recommendation system in machine learning and data mining to filter the unseen information and predict the users likes and dislikes.

Algorithm Advantages

- The algorithm performs well for multiclass prediction.
- The prediction of class on test data is faster and it is also easy to apply.
- A Naïve Bayes classifier performs better when compared to other classifiers like e.g. Logistic regression when the assumption of independence holds, it also requires less amount of training data.
- The algorithm performs better for categorical input variable(s) when compared to numerical variable(s). The numerical variable assumes a normal distribution.

Algorithm Disadvantages

- The model will be unable to make a prediction if the categorical variable has a category in test data, which was not available in training data.
- In such a case the model assigns a zero (0) probability. This is called as "Zero Frequency". This problem can be overcome with a smoothing technique. One of the simplest method used for smoothing is called Laplace estimation.
- The assumption of independent predictors in Naïve Bayes algorithm is a limitation as in real world it is almost impossible to get a set of predictors which are independent.

Syllabus Topic : Support Vector Machines (SVM)

4.8 Introduction to Support Vector Machine (SVM)

- Support vector Machine is another method used for classification.
- It can classify both Linear as well as Non Linear Data.

- The objective of SVM is to find a hyperplane (A decision boundary separating the tuples of one class from another) in an N - dimensional space (where N represents the number of features) that distinctly classify the data points.

E.g.

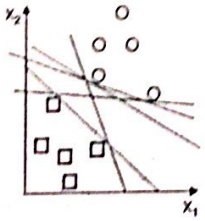


Fig. 4.8.1 : All possible Hyperplanes to classify the data points

- As we can see in the above diagram, there are many possible hyperplanes that could be chosen, but the main objective is to find a plane that has maximum margin i.e. maximum distance between data points of both classes as shown in the Fig. 4.8.2.
- Maximizing provides reinforcement so that future data points can be classified with more confidence.

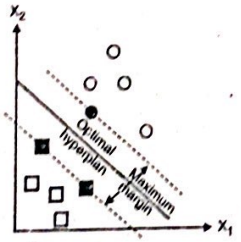


Fig. 4.8.2 : Optimal hyperplane having maximum margin

Working of Support Vector Machine

- SVM uses a Non Linear mapping to transform the original training data into a higher dimension.
- In this dimension it searches for a linear optimal separating hyperplane.
- It finds this hyperplane using support vectors (essentially training tuples) and margins (defined by support vectors).
- In SVM, if the output of the Linear function is greater than 1, it is identified as one class and if the output is -1 it is identified with another class.

Hyperplanes

- Hyperplanes are decision boundaries that are used to classify the data points.
- The dimension of the hyperplane depends depends upon the number of features.
  - Number of Input features is 2 : then hyperplane is a Line
  - Number of Input features is 3 : the hyperplane is a two dimensional plane
- Difficult to imagine a hyperplane when input features are more than 3.

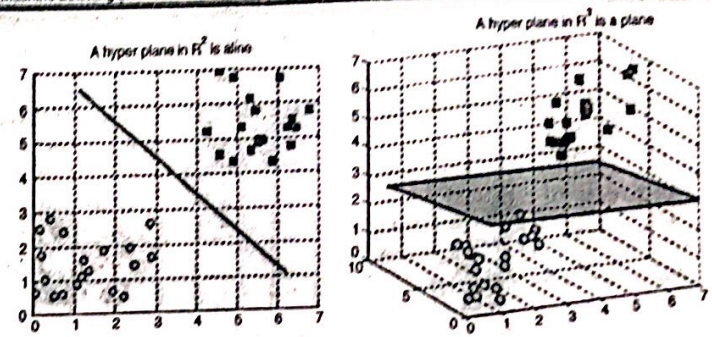


Fig. 4.8.3 : Hyperplane as a Line and a two dimensional plane

Support Vectors

- Support vectors are data points that are closer to hyperplane.
- These support vectors are used to maximize the margin of the classifier.
- Deletion of these support vectors will change the position of the Hyperplane.

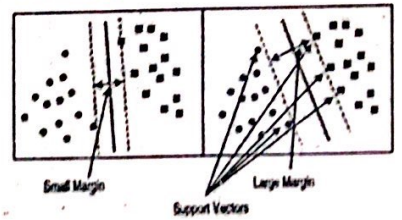


Fig. 4.8.4 : Support vectors and Margin separating the data points

4.8.1 Tuning Hyperparameters

Kernel

- Kernel is used to transform the given data into the required form.
- Different types of kernel functions may be used in Support vector machine, such as Linear, Polynomial, Radial Basis function and sigmoid kernel.
- Polynomial and RBF kernels compute the separation line in higher dimension, they are also useful for non-linear hyperplane.
- For separating the classes that are either curved or non-linear, more complex kernels can be used.

Regularization

- A penalty parameter called as C is used in Scikit-learn library as a regularization parameter.

- The term C represents misclassification or error term, which tells the SVM optimization by how much the error is bearable. This is how the trade-off between the decision boundary and misclassification term is controlled.
- A small-margin hyperplane is created by a small value of C and a larger value of C creates a larger-margin hyperplane.

Gamma

- The value of gamma decides the number of data points in the calculation of separating line.
- A low value of gamma will consider only nearby points whereas a high value of gamma will consider all the data points in the calculation.
- A low value of gamma will loosely fit the training data, whereas a high value will overfit the training data.

Advantages of SVM

- SVM is a highly accurate method due to the ability of modeling complex non-linear decision boundaries.
- When compared to other methods, SVM is less prone to overfitting.
- The support vectors of the model provide a description of the learned model.
- SVM can be used for prediction (numeric) and also as a classifier.

Disadvantages

- The training time of even the fastest SVMs can be extremely slow on large data sets.
- Less effective on noisier datasets with overlapping classes.

Applications

- Handwritten digit recognition.
- Object recognition.
- Speaker identification.
- Benchmark time-series prediction tests.

Syllabus Topic : Linear Support Vector Machines

4.9 Linear Support Vector Machines

- Let us consider a two class problem where the classes are linearly separable.
- Consider D as the data set which can be given by

  $(X_1, y_1), (X_2, y_2), \ldots, (X_{|D|}, y_{|D|})$,

  Where X_i is the set of training tuples and y_i the associated class labels which can take + 1 or – 1 as its values.
- The Linearly separable classes as shown in the Fig. 4.9.1.

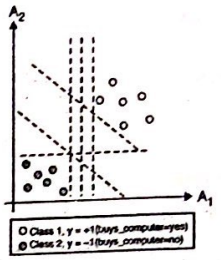


Fig. 4.9.1 : Linearly Separable Data

- Goal here is to find the best possible separating line (considering data as 2D) having minimum classification error on previously unseen tuple.
- Incase if the data is 3D, the goal would be to find the best separating hyperplane with minimum classification error.
- So to Generalise the concept for N dimensions, the Goal is to find the best Hyperplane.
- Support Vector Machine works by finding the maximum marginal hyperplane.

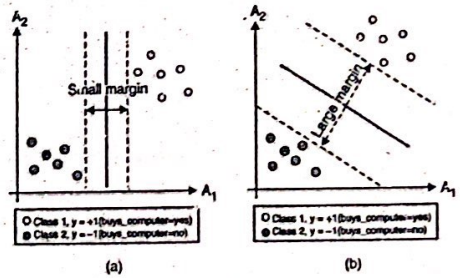


Fig. 4.9.2 : Two possible hyperplanes separating the data

- From Fig. 4.9.2 we can see that there are two possible hyperplanes separating the data, one with smaller margin and the other with larger margin.
- The hyperplane with larger margin can classify the data more better in the future as compared to the smaller margin hyperplane.
- This is the reason SVM searches the hyperplane with largest margin, that is Maximal Margin Hyperplane (MMH).

Definition of Margin : It is the shortest distance from a hyperplane to one side of its margin is equal to the shortest distance from the hyperplane to the other side of its margin, where the "sides" of the margin are parallel to the hyperplane.

- A separating hyperplane can be given as

$$W \cdot X + b = 0 \quad \ldots(4.9.1)$$

W : is a weight vector, $W = \{w_1, w_2, w_3, \ldots w_n\}$

n : is the number of attributes

b : is a scalar (called as bias)

- To understand the classification let us consider two input attributes A_1 and A_2
- Training tuples are 2D, $(X = (x_1, x_2))$ where x_1 and x_2 are attributes of A_1 and A_2
- Considering w_0 as the additional weight, and rewriting Equation (4.9.1) as

$$w_0 + w_1 x_1 + w_2 x_2 = 0 \quad \ldots(4.9.2)$$

- Any point that lies above the hyperplane satisfies the following equation

$$w_0 + w_1 x_1 + w_2 x_2 > 0 \quad \ldots(4.9.3)$$

- And any point that lies below the hyperplane satisfies the following equation

$$w_0 + w_1 x_1 + w_2 x_2 < 0 \quad \ldots(4.9.4)$$

- Adjusting the weight, and hyperplanes defining the sides of the margin can be written as

$$H_1 : w_0 + w_1 x_1 + w_2 x_2 \geq 1 \quad \text{for } y_i = +1, \quad \ldots(4.9.5)$$

$$H_2 : w_0 + w_1 x_1 + w_2 x_2 \leq -1 \quad \text{for } y_i = -1 \quad \ldots(4.9.6)$$

- From the above two equations (4.9.5) and (4.9.6) we can say that , any tuple that falls on or above H_1 belongs to class + 1 and any tuple that falls on or below H_2 belongs to class -1.
- Combining the two equations (4.9.5) and (4.9.6) we get

$$Y_i (w_0 + w_1 x_1 + w_2 x_2) \geq 1, \quad \forall i \quad \ldots(4.9.7)$$

- Any training tuples that fall on hyperplanes H_1 or H_2 (i.e., the "sides" defining the margin) satisfy Eq. (4.9.7) and are called support vectors. That is, they are equally close to the (separating) MMH.

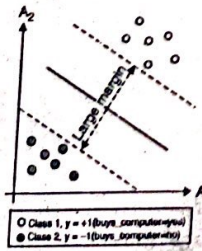


Fig. 4.9.3 : Support vectors with thick boundary

- The distance from the separating hyperplane to any point on H_1 is $1/\|W\|$, where $\|W\|$ is the Euclidean norm of $\sqrt{W}$ W. By definition, this is equal to the distance from any point on H_2 to the separating hyperplane. Therefore, the maximal margin is $2/\|W\|$.

Syllabus Topic : Scikit-Learn Implementation

4.10 Scikit-Learn Implementation

- A Support vector machine(SVM) is a type of supervised machine learning classification algorithm used for classification, regression and outlier detection.
- The support vector machines in scikit-learn support both dense and sparse sample vectors as input.
- SVC, NuSVC and LinearSVC are classes capable of performing multiclass classification on a dataset.
- SVC and NuSVC implements "one-against-one" approach for classification , on the other hand LinearSVC implements "one-vs-the-rest" multiclass strategy, thus training n class models. Incase of only two classes only one model is trained.

Syllabus Topic : Linear Classification

4.10.1 Linear Based Classification

E.g. In Scikit-learn, a simple classification task in which two classes of points are well separated.

Standard Imports

```
%matplotlib inline
import numpy as np import matplotlib.pyplot as plt
from scipyimport stats
# use seaborn plotting defaults
import seaborn as sns; sns.set()

from sklearn.datasets.samples_generator import make_blobs
X, y = make_blobs(n_samples=50, centers=2,random_state=0, cluster_std=0.60) plt.scatter(X[:, 0], X[:, 1], c=y, s=50,
cmap='autumn');
```

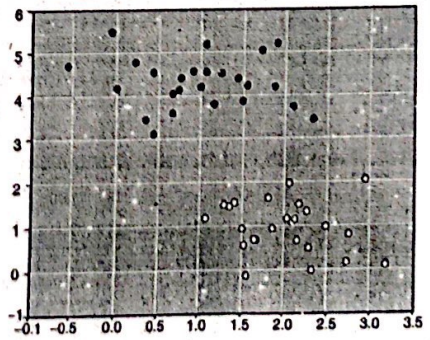


Fig. 4.10.1 : Plot showing data which is linearly separable

Draw Lines that can divide the data into two separate classes

```
xfit=np.linspace(-1, 3.5)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
plt.plot([0.6], [2.1], 'x', color='red', markeredgewidth=2, markersize=10) for m, b in [(1, 0.65), (0.5, 1.6), (-0.2, 2.9)]:
plt.plot(xfit, m *xfit + b, '-k')
```

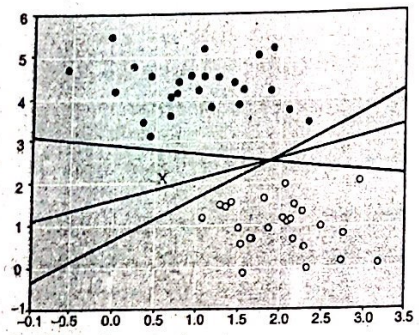


Fig. 4.10.2 : Plot showing Lines separating the data

Rather than simply drawing a zero-width line between the classes, we can draw around each line a *margin* of some width, up to the nearest point.

Fitting a support vector machine

Use Scikit-Learn's support vector classifier to train an SVM model on this data.

```
from sklearn.svm import SVC
# "Support vector classifier"
model = SVC(kernel='linear', C=1E10)
model.fit(X, y)
SVC(C=10000000000.0, cache_size=200, class_weight=None, coef0=0.0,  decision_function_shape=None, degree=3,
gamma='auto', kernel='linear',  max_iter=-1, probability=False, random_state=None, shrinking=True,  tol=0.001,
verbose=False)

def plot_svc_decision_function(model, ax=None, plot_support=True):
"""Plot the decision function for a 2D SVC"""
if ax is None:
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()
# create grid to evaluate model
x = np.linspace(xlim[0], xlim[1], 30)
y = np.linspace(ylim[0], ylim[1], 30)
Y, X = np.meshgrid(y, x)
```



```
xy = np.vstack([X.ravel(), Y.ravel()]).T
P = model.decision_function(xy).reshape(X.shape)
# plot decision boundary and margins
ax.contour(X, Y, P, colors='k',
           levels=[-1, 0, 1], alpha=0.5,
           linestyles=['--', '-', '--'])
# plot support vectors
if plot_support:
    ax.scatter(model.support_vectors_[:, 0],
               model.support_vectors_[:, 1],
               s=300, linewidth=1, facecolors='none');
    ax.set_xlim(xlim)
    ax.set_ylim(ylim)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
plot_svc_decision_function(model);


model.support_vectors_
```

Output

```
array([[ 0.44359863,  3.11530945],       [ 2.33812285,  3.43116792],       [ 2.06156753,  1.96918596]])
```

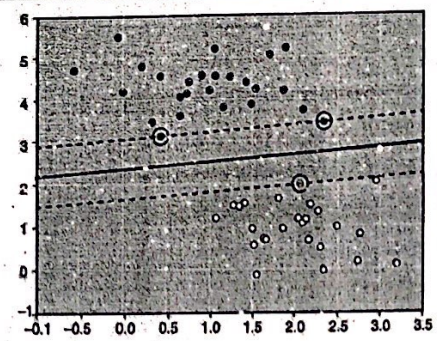


Fig. 4.10.3 : Support vectors

Syllabus Topic : Kernel Based Classification

4.10.2 Kernel Based Classification

- Some problems cannot be solved using Linear hyper plane for e.g. as shown below.
- In this type of cases, SVM uses a kernel trick to transform the input space to a higher dimensional as shown in the Fig. 4.10.4(b).

- The data points are plotted on x-axis and z-axis, now these points can be easily segregated.

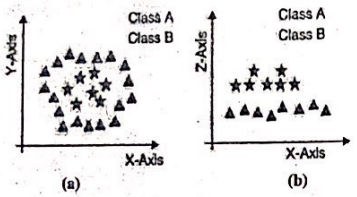


Fig. 4.10.4

- The kernel takes a low dimensional space and transforms it into higher dimensional space.
- It converts a non-separable problem into a separable problem by adding more dimensions to it.
- This is most useful in non-linear separation problem.
- Following are the different kernels used in SVM :

☞ **Linear Kernel**

- A Linear kernel is a dot product of any two given observations.
- The product between two vectors is the sum of the multiplication of each pair of input values.

☞ **Polynomial Kernel**

- A more generalized form of Linear kernel is Polynomial kernel.
- The polynomial kernel can differentiate between curved or nonlinear input space.

☞ **Radial Basis Function Kernel**

- The input space can be mapped in infinite dimensional space by RBF kernel.
- It is a local kernel and can create complex regions within the feature space like e.g. closed polygons in 2D space.

☞ **Sigmoid Kernel**

- The sigmoid kernel has its origin from Neural networks.
- Neural network approach as we know is used for classification of input data.

☞ **Kernel Functions**

$$K(X_i, X_j) = \begin{cases} X_i \cdot X_j & \text{Linear} \\ (\gamma X_i \cdot X_j + C)^d & \text{Polynomial} \\ \exp(-\gamma |X_j - X_j|^2) & \text{RBF} \\ \tanh(\gamma X_i \cdot X_j + C) & \text{Sigmoid} \end{cases}$$

where $K(X_i, X_j) = \phi(X_i) \cdot \phi(X_j)$

Syllabus Topic : Non - Linear Examples

4.10.3 Non Linear Examples

- A Non Linear example

```
from sklearn.datasets.samples_generator import make_circles
X, y = make_circles(100, factor=.1, noise=.1)
clf = SVC(kernel='linear').fit(X, y)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
plot_svc_decision_function(clf, plot_support=False);
```

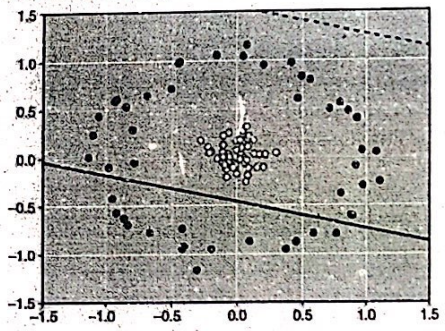


Fig. 4.10.5 : Non Linear separable data

- From the above figure it is clear that the data is not linearly separable, for this we can use a kernel. (e.g. radial basis function).

```
r = np.exp(-(X ** 2).sum(1))
```

- Lets visualize this extra dimension using a three dimensional plot

```
from mpl_toolkits import mplot3d
def plot_3D(elev=30, azim=30, X=X, y=y):
ax = plt.subplot(projection='3d')
ax.scatter3D(X[:, 0], X[:, 1], r, c=y, s=50, cmap='autumn')
ax.view_init(elev=elev, azim=azim)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('r')
interact(plot_3D, elev=[-90, 90], azim=(-180, 180),
X=fixed(X), y=fixed(y));
```

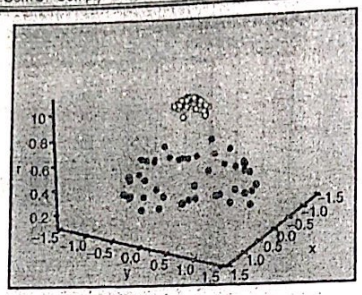


Fig. 4.10.6 : Linearly separable data using radial kernel

- The kernel model hyperparameter in Scikit-learn can be used for applying kernalized SVM. i.e. for changing the Linear kernel to RBF or some other kernel.

```
clf= SVC(kernel='rbf',C=1E6) clf.fit(X, y)

SVC(C=1000000.0, cache_size=200, class_weight=None, coef0=0.0,  decision_function_shape=None, degree=3,
gamma='auto', kernel='rbf',  max_iter=-1, probability=False, random_state=None, shrinking=True,  tol=0.001,
verbose=False)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
plot_svc_decision_function(clf)
plt.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1],          s=300, lw=1, facecolors='none');
```

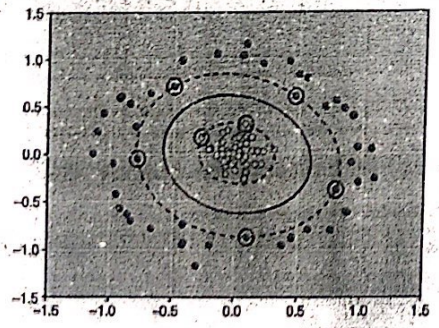


Fig. 4.10.7

Syllabus Topic : Controlled Support Vector Machines

4.11 Controlled Support Vector Machines

- While working with real data sets, SVM classifier can result in many support vectors, which can slow down the whole process.

- To find a tradeoff between precision and the number of support vectors, scikit learn implementation provides NuSVC implementation where the parameter nu(bounded between 0 – not included and 1) can be used to control the number of support vectors.

Syllabus Topic : Support Vector Regression

4.12 Support Vector Regression

- The method of Support Vector machines can be extended to solve regression problems. This is called as Support Vector Regression.
- As discussed above the model produced by Support Vector machine for classification uses only a subset of training data, as the cost function for building the model does not consider the data that lie beyond the margin.
- Similarly the model produced by Support vector regression also depends on a subset of training data as the cost function for building the model ignores any data close to the model prediction.
- In Scikit-learn, there are three different implementations of Support vector regression SVR, NuSVR and LinearSVR.
- LinearSVR is faster than SVR but only considers Linear Kernels.
- NuSVR implements slightly different from SVR and LinearSVR.
- E.g. of SVR in Scikit-learn

```
from sklearn import svm
>>> X = [[0, 0], [2, 2]]
>>> y = [0.5, 2.5]
>>> clf = svm.SVR()
>>> clf.fit(X, y)
SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='auto_deprecated', kernel='rbf', max_iter=-1,
shrinking=True, tol=0.001, verbose=False)
>>> clf.predict([[1, 1]]) array([1.5])
```

Review Questions

Q. 1 Explain Bayes Theorem.

Q. 2 Explain Naïve Bayes classifier.

Q. 3 Explain the different variants of Naïve Bayes in scikit learn library.

Q. 4 Explain Support vector Machines.

Q. 5 Explain Linear support vector machine and its implementation in Scikit learn.

Q. 6 Explain Linear and kernel based classification in Support vector machines.

Q. 7 Explain Support vector regression.

Chapter Ends...

□□□

UNIT V

CHAPTER 5

Decision Trees and Ensemble Learning

Syllabus Topics

Decision Trees - Impurity measures, Feature Importance. Decision Tree Classification with Scikit-learn, Ensemble Learning-Random Forest, AdaBoost, Gradient Tree Boosting, Voting Classifier.

Clustering Fundamentals - Basics, K-means: Finding optimal number of clusters, DBSCAN, Spectral Clustering. Evaluation methods based on Ground Truth- Homogeneity, Completeness, Adjusted Rand Index.

Introduction to Meta Classifier : Concepts of Weak and eager learner, Ensemble methods, Bagging, Boosting, Random Forests.

Syllabus Topic : Decision Trees – Impurity Measures

5.1 Decision Trees

Suppose a database D is given as $D = \{ t_1, t_2, \ldots, t_n \}$ and a set of desired classes are $C = \{ C_1, \ldots, C_m \}$, the **Classification problem** is to define the mapping m in such a way that which tuple of database D belongs to which class of C. Actually divides D into **equivalence classes**.

Classification Example

How teacher gives grades to students based on their marks obtained:

If $x >= 90$ then grade = A.

If $80 <= x < 90$ then grade = B.

If $70 <= x < 80$ then grade = C.

If $60 <= x < 70$ then grade = D.

If $x < 60$ then grade = E.

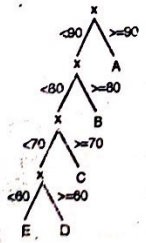


Fig. 5.1.1 : Classification of grading

Decision tree learning is appropriate for problems having characteristics given below:

- Instances are represented by a fixed set of attributes (e.g. Gender) and their values (e.g. male, female) described as **attribute-value pairs**.
- If the attribute has small number of disjoint possible values (e.g. high, medium, low) or there are only two possible classes (e.g. true, false) then decision tree learning is easy.

- Extension to decision tree algorithm also handles real valued attributes (e.g. salary).
- Decision tree gives a class label to each instance of the dataset.
- If some of the examples have unknown values, still decision tree method can be used (Example: Wind value is known for few tuples).
- Decision trees or if-then rules represent learned functions which improve readability.

Decision Tree Representation

- Decision tree has leaf nodes and decision nodes.
- Every branch's last node is a leaf node which represents the class label.
- Decision node is used to take decision for the class label after some trials. It has a leaf node or sub tree.
- Play Tennis example for representing decision Trees.

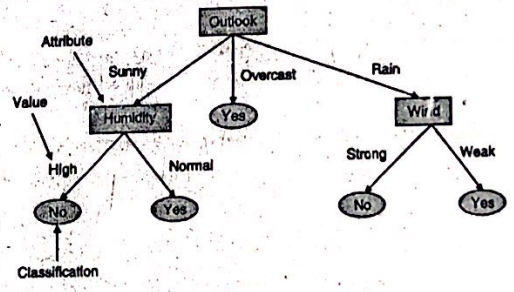


Fig. 5.1.2 : Representation of decision tree

Binary Decision tree

- It is a structure based on a sequential decision process.
- It goes from the root and based on evaluating features one of the two branches is chosen.
- Repeat the same procedure until a final leaf node or classification target is reached.

5.1.1 Impurity measures

1. Gini Impurity Index

- Suppose all attributes are continuous-valued.
- Assume that each value of an attribute has many possible splits.
- It can be adapted for categorical attributes.
- Gini is used in CART (Classification and Regression Trees), IBM's IntelligentMiner system, SPRINT (Scalable PaRallelizable INduction of decision Trees).
- If a data set T contains examples from n classes, Gini index, Gini (T) is defined as

$$gini(T) = 1 - \sum_{j=1}^{n} p_j^2$$

- Where, p_j is the relative frequency of class j in T.
- ginif(T) is minimized if the classes in T are skewed.
- Gini index of split is defined below after split T1 and T2 of sizes N1 and N2 respectively.

$$gini_{split}(T) = \frac{N_1}{N} gini(T_1) + \frac{N_2}{N} gini(T_2)$$

- For every attribute, each of the possible binary splits is considered. For a discrete-valued attribute, the attribute providing smallest $gini_{split}(T)$ is picked out to split the node. For continuous-valued properties, each possible split-point must be believed.
- The Gini impurity measures the probability of a misclassification if a label is randomly selected using the probability distribution of the branch. If all the samples belong to a single category, then index reaches its minimum (0.0).

2 Cross-entropy impurity index

- **Cross-entropy impurity index** is based on Information theory.
- When all the samples belong to one class are represented in a split, then null values are assumed. But if there is uniform distribution, then it is maximum.
- It permits to select the split which actually minimizes uncertainty about classification.

$$\text{Entropy} = -\sum_j p_j \log_2 p_j$$

- Information Gain (IG), determine **which attribute** in a given set of training feature vectors is **most useful**. We will use it to decide the ranking of attributes in the nodes of a decision tree.

The Information Gain (IG) can be defined as follows:

- All attributes are considered to be categorical.
- It can be adapted for continuous-valued attributes.
- The attribute which has the highest information gain is selected for split.
- Assume there are two classes, P and N.
- Consider S samples, out of these p samples belongs to class P and n samples belongs to class N.
- The amount of information, needed to decide if random example in S belongs to P or N is defined as

$$I(p, n) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$

- Assume that using attribute A, a set S will be partitioned into sets $\{S_1, S_2, \ldots, S_v\}$.
- If S_i contains p_i examples of P and n_i examples of N, the entropy, or the expected information needed to classify objects in all sub trees S_i is

$$E(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p+n} I(p_i, n_i)$$

Entropy (E)

- The expected amount of information (in bits) needed to assign a class to a randomly drawn object in S under the optimal, shortest-length code.
- **Calculate information gain i.e. gain (A) :** Measures reduction in entropy, achieved because of the split. Take the split that achieves most reductions (maximizes GAIN)

$$\text{Gain}(A) = I(p, n) - E(A)$$

3. Misclassification impurity index

- The misclassification impurity is the simplest index, defined as :

$$I_E = 1 - \max\{p(i|t)\},$$

- It is a useful criterion for pruning but not recommended for growing a decision tree since it is less sensitive to changes in the class probabilities of the nodes.

Syllabus Topic : Feature Importance

5.1.2 Feature Importance

- To predict the output value, feature importance helps to evaluate significance of every feature of multi-dimensional dataset.
- Decision trees offer a different approach based on the impurity reduction determined by every single feature.
- In particular, considering a feature x_i, its importance can be determined as :

$$\text{Importance}(x_i) = \sum_k \frac{N_k}{N} \Delta I_{x_i}$$

- The sum is extended to all nodes where x_i is used, and N_k is the number of samples reaching the node k.

Sample code for feature importance in python

- Considering the iris example and converting to a pandas.DataFrame()

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
import pandas as pd

clf = DecisionTreeClassifier(random_state=0)
iris = load_iris()
iris_pd = pd.DataFrame(iris.data, columns=['sepal_length', 'sepal_width', 'petal_length', 'petal_width'])
clf = clf.fit(iris_pd, iris.target)
print(dict(zip(iris_pd.columns, clf.feature_importances_)))

Output :- Showing the feature importance of each feature
{'sepal_length': 0.0, 'sepal_width': 0.01333333333333329, 'petal_length': 0.06405595813204505, 'petal_width':
0.9226107085346216}
```

Syllabus Topic : Decision Tree Classification with Scikit-learn

5.1.3 Decision Tree Classification with Scikit-learn

Consider iris dataset , using the DecisionTreeClassifier class of Scikit-learn for binary decision tree.

```
from sklearn.datasets import load_iris
from sklearn import tree
iris = load_iris()
clf = tree.DecisionTreeClassifier()
clf = clf.fit(iris.data,iris.target)
# export the tree in Graphviz format using the export_graphviz
import graphviz
dot_data = tree.export_graphviz(clf,out_file=None)
graph = graphviz.Source(dot_data)
graph.render("iris")

dot_data = tree.export_graphviz(clf,out_file=None,
feature_names = iris.feature_names,
class_names = iris.target_names,
filled = True,rounded=True,
special_characters=True)
graph = graphviz.Source(dot_data)
graph
```

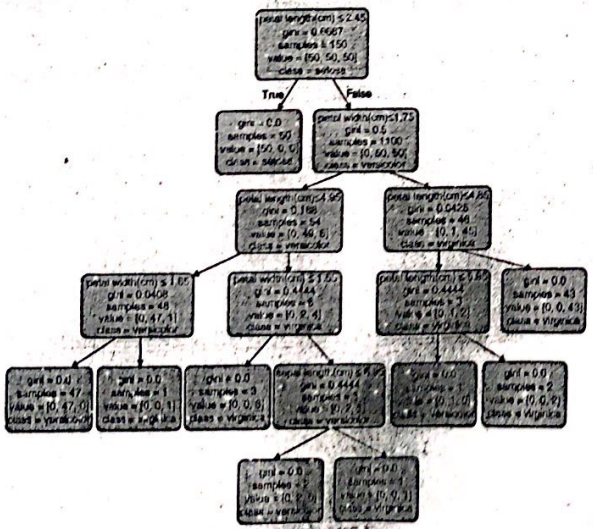


Fig. 5.1.3 : Graph of iris example

Syllabus Topic : Ensemble Learning

5.1.4 Ensemble Learning

Ensemble learning combines various set of learners (individual models) together which actually improvise on the stability and predictive power of the model.

- Combining classifier is an ensemble method which increases the accuracy.
- To get new improved model M^*, combine a series of n learned models, $M_1, M_2, ..., M_n$

Popular ensemble methods are :

1. Bagged (or Bootstrap) trees

- Base learners are generated in parallel, so it is a parallel ensemble method (e.g. Random Forest).
- The basic motivation of parallel methods is to **exploit independence between the base learners** since the error can be reduced dramatically by averaging.
- For example, we can train M different trees on different subsets of the data (chosen randomly with replacement) and compute the ensemble :

$$f(x) = 1/M \sum_{m=1}^{M} f_m(x)$$

- In generalized bagging, different learners can be used in different population to reduce the variance error.

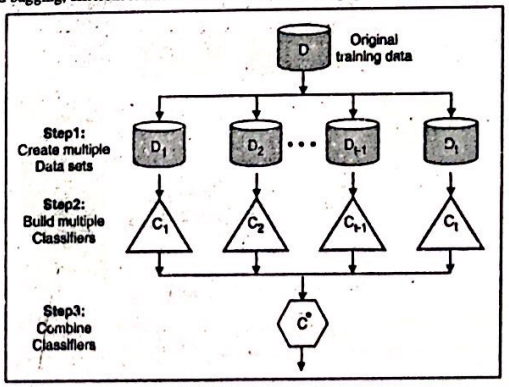


Fig. 5.1.4 : Bagging or bootstrap

- It averages the prediction from the accumulation of various classifiers used.
- A bootstrap method is used, for data set D of n tuples, for each iteration n tuple are sampled with replacement from D.
- In every iteration , a classifier model M is learned from training data set
- For unknown sample Y, each classifier gives class prediction.

- The bagged classifier M* uses the voting method i.e. the sample tuple Y is assigned the class with the most votes to tuple Y.
- For continuous values, it can be used for prediction by taking the average of all predictions for a given sample.

2. Boosted trees

- It is a family of algorithms that are able to convert weak learners to strong learners.
- It is a sequential ensemble method where the base learners are generated sequentially (e.g. AdaBoost).
- The basic motivation of sequential methods is to **exploit the dependence between the base learners**. The overall performance can be boosted by weighing previously mislabelled examples with higher weight.
- In boosting, each training tuple has weight.
- n number of classifiers are learned iteratively.
- After learning of M_i classifier, every time the weights are updated for next classifier learning i.e. M_{i+1}. So if the tuples which were misclassified by M_i will get higher weight for next classifier.
- Use voting method, where check the votes of each classifier to get the final M* which helps to get the accuracy.
- The extended boosting algorithm works for the prediction of continuous values.
- Boosting tends to accomplish greater accuracy as compared to bagging, there is a risk of overfitting the model.

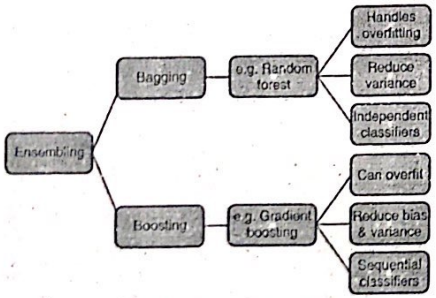


Fig. 5.1.5 : Ensemble Learning

Syllabus Topic : Random Forest

5.1.4(A) Random Forest

- It is a supervised classification algorithm.
- It creates the forest with a number of trees as the name suggests.
- If the number of trees in the forest are more, the accuracy result is high.
- Random forest handles the missing values.
- Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features.

- The algorithm selects random subset of features to split the node.

Random Forest pseudocode

1. Select "k" features randomly from total "m" features where $k << m$.
2. Using the bet split point, calculate the node d from selected k features.
3. Using the best split, split the node d into child nodes.
4. Repeat 1 to 3 steps until "l" number of nodes has been reached.
5. Build forest by repeating steps 1 to 4 for "n" number times to create "n" number of trees.

Random forest prediction pseudo code

1. Predict and store the outcome using the rules of each randomly created decision tree.
2. Count on the votes for each predicted target.
3. The last prediction is selected by taking the high voted predicted outcome.

With two trees, you can see how a random forest would look like

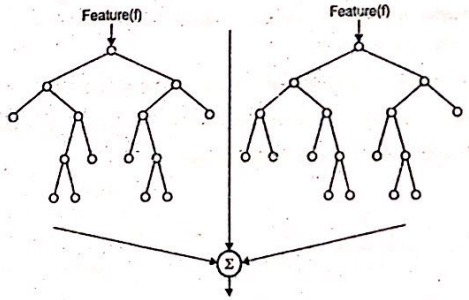


Fig. 5.1.6 : Random forest with 2 trees

Advantages

- It can be used for both regression and classification tasks.
- It is a very handy and easy to use algorithm as it's default hyper parameters often produce a good prediction result.
- The classifier won't overfit the model if there are enough trees in the forest.

Disadvantages

- Many trees are generated which makes algorithm slow and not suited for real time prediction.
- It's not a descriptive tool, but only predictive model.
- If dataset is noisy , then random forest may overfit.
- For data, including categorical variables with different number of levels, random forests are biased in favour of those attributes with more levels. Therefore, the variable importance scores from random forest are not reliable for this type of data.

Syllabus Topic : Ada Boost

5.1.4(B) Ada Boost

- It combines weak classifier algorithm to form strong classifier, then by selecting, training set at every iteration multiple classifiers are combined which gives good accuracy score.
- Correct amount of weight can be assigned in final voting, which improves accuracy.

Algorithm

Step 1 : Choose the training set and train the algorithm.

Step 2 : Retrains the algorithm iteratively by selecting another set of training data based on the accuracy of previous training.

Step 3 : The weight-age depends on the accuracy achieved for each trained classifier.

Step 4 : It assigns weight to each training item.

Step 5 : Higher weights are assigned to misclassified item so that those items can appear in the training subset of next classifier with higher probability.

Step 6 : After training weights is assigned to each classifier also based on accuracy.

Step 7 : Higher weights are assigned to more accurate classifier to get more impact on the final outcome.

Step 8 : The mathematical formula for Adaboost is given below.

Where,

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

T : Number of classifiers

$h_t(x)$: Output of weak classifier t for input x

α_t : Weight assigned to the classifier.

α_t is calculated as follows:

$\alpha_t = 0.5 * \ln((1-E)/E)$ where E is error rate.

Step 9 : Initially, all the input training examples has equal weightage. The mathematical formula to update the weight of each training example is given below.

$$D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

Where,

D_t : weight at previous level.

Z_t : sum of all weights used to normalize the weights.

y_i is y par of training example (xi. yi) y coordinate for simplicity.

Syllabus Topic : Gradient Tree Boosting

5.1.4(C) Gradient Tree Boosting

- Gradient tree boosting is a technique that allows you to build a tree ensemble step by step with the goal of minimizing a target loss function. Gradient boosting has main three components:
  1. A loss function to be optimized.
  2. A weak learner to make predictions.
  3. An additive model to add weak learners to minimize the loss function.
- Mean squared error (MSE) as loss defined as:

$$\text{Loss} = \text{MSE} = \Sigma (y_i - y_i^P)^2$$

Where, y_i = i^{th} target value, y_i^P = i^{th} prediction, $L(y_i, y_i^P)$ is loss function

- MSE should be minimum for our predictions.
- Use a gradient descent and update our predictions based on a learning rate, we can find the values where MSE is minimized.

$$y_i^P = y_i^P + \alpha * \delta \Sigma (y_i - y_i^P)^2 / \delta y_i^P$$

Which becomes,

$$y_i^P = y_i^P - \alpha * 2 * \Sigma (y_i - y_i^P)$$

Where, α is learning rate and $\Sigma (y_i - y_i^P)$ is the sum of residuals.

Steps of Gradient Boost algorithm

Step 1 : Assume mean is the prediction of all variables.

Step 2 : For each observation calculate errors using mean (latest prediction).

Step 3 : Find the variable that can split the errors perfectly and find the value for the split. This is assumed to be the latest prediction.

Step 4 : Calculate errors of each observation from the mean of both the sides of the split (latest prediction).

Step 5 : Repeat the step 3 and 4 till the objective function maximizes/minimizes.

Step 6 : Take a weighted mean of all the classifiers to come up with the final model.

Syllabus Topic : Voting Classifier

5.1.4(D) Voting Classifier

Voting classifier has two strategies

1. **Hard Voting :** Final class label is predicted based on the most frequent class label of classification models.
2. **Soft Voting :** Final class label is predicted by averaging the class probabilities.

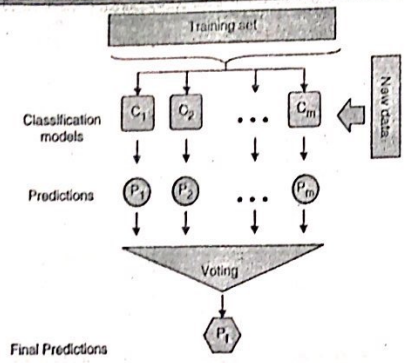


Fig. 5.1.7 : Voting classifier block diagram

Hard Voting or Majority Voting

Predict the class label Y via majority voting of each classifier C_j:

$$Y = \text{mode}\{C_1(x), C_2(x), ..., C_m(x)\}$$

Assume 3 classifiers and predicted class of each as given below :

classifier 1 > class 0

classifier 2 > class 0

classifier 3 > class 1

$$y = \text{mode}\{0,0,1\} = 0$$

So final predicted class for the sample is "0"

Weighted Majority Vote

Weighted majority vote can be calculated by associating a weight w_j with classifier C_j :

$$\hat{y} = \arg\max_i \sum_{j=1}^{m} w_j X_A (C_j(x) = i),$$

where X_A is the characteristic

A is the set of unique class labels.

Assume 3 classifiers and predicted class of each as given below :

classifier 1 > class 0

classifier 2 > class 0

classifier 3 > class 1

Assign the weights as {0.2, 0.2, 0.6} respectively for each class.

Prediction for Y is

$$\arg\max_i [0.2 \times i_0 + 0.2 \times i_0 + 0.6 \times i_1] = 1$$

Soft Voting

Class label is predicted based on the predicted probabilities p for the classifier. This works only if classifiers are well-calibrated.

$$\hat{y} = \arg\max_i \sum_{j=1}^{m} w_j p_{ij},$$

where W_j is the weight that can be assigned to the jth classifier.

Assume the same example with binary classification having class labels {0 , 1} our ensemble could make the following prediction :

C1(x) → [0.9,0.1]

C2(x) → [0.8,0.2]

C3(x) → [0.4,0.6]

Using uniform weights, we compute the average probabilities :

$$p(i_0 \mid x) = \frac{0.9 + 0.8 + 0.4}{3} = 0.7$$

$$p(i_1 \mid x) = \frac{0.1 + 0.2 + 0.6}{3} = 0.3$$

$$\hat{y} = \arg\max_i [p(i_0 \mid x), p(i_1 \mid x)] = 0$$

However, assigning the weights {0.1, 0.1, 0.8} would yield a prediction $\hat{y} = 1$:

$$p(i_0 \mid x) = 0.1 \times 0.9 + 0.1 \times 0.8 + 0.8 \times 0.4 = 0.49$$

$$p(i_1 \mid x) = 0.1 \times 0.1 + 0.2 \times 0.1 + 0.8 \times 0.6 = 0.51$$

$$\hat{y} = \arg\max_i [p(i_0 \mid x), p(i_1 \mid x)] = 1$$

Syllabus Topic : Clustering Fundamentals - Basics

5.2 Clustering Fundamentals

5.2.1 Basics of Clustering

- Clustering is an unsupervised learning problem.
- No classes or groups are known earlier.
- Data is divided into different groups, called as clusters.
- Clusters should be made based on data objects having high inter similarity and low intra similarity.
- In hard clustering techniques, each element must belong to a single cluster and in soft clustering or fuzzy clustering, is based on a membership score that defines how much the elements are "compatible" with each cluster.

- Graphical representation can be shown :

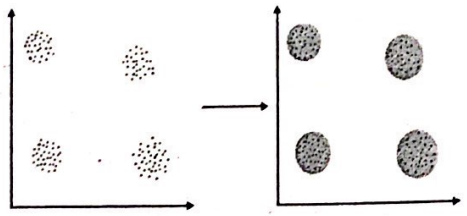


Fig. 5.2.1 : Clustering Graphical example

- From Fig. 5.2.1, we can observe there are 4 clusters based on their geometrical distance grouped together is called distance based clustering.
- Conceptual clustering is based on common or descriptive concepts of data objects.

Applications

Clustering algorithms can be applied in many disciplines :

- **Marketing :** Clustering can be used for targeted marketing. For e.g. Given a customer database, containing properties and past buying records. Similar groups of customers can be identified and grouped into one cluster.
- **Biology :** Clustering can also be employed in classifying plants and animals into different categories based on their characteristics.
- **Libraries :** books can be clusters based on their details.
- **Insurance :** Different groups of policy holders can be placed with clustering. For e.g. identify policy fraud or customers with high claim.
- **City-planning :** Groups of houses based on location and type.
- **Earthquake studies :** Clustering can also be used to identify dangerous zones based on earthquake epicentre.
- **WWW :** Clustering can be used to find groups of similar access patterns using weblog data. It can also be used for classification of documents.

Syllabus Topic : K-means

5.2.2 K-means

- In 1967, J. MacQueen and then in 1975 J. A. Hartigan and M. A. Wong developed K-means clustering algorithm.
- In k-means, k is the number of clusters given by user and objects are classified into k clusters based on their attributes.
- K-means is one of the simplest unsupervised learning algorithms.
- Define K centroids for K clusters which are generally far away from each other.
- Group the objects into clusters based on the distance with respect to centroid

- After this first step, again calculate the new centroid for each cluster based on the elements of that cluster.
- Follow the same method and group the elements based on new centroid.
- At every step, the centroid changes and elements move from one cluster to another.
- Do the same process till no element is moving from one cluster to another i.e. till two consecutive steps with the same centroid and the same elements are obtained.
- Finally, this algorithm aims at minimizing an *objective function*, in this case a squared error function. The objective function is given below,

$$J = \sum_{j=1}^{k} \sum_{i=1}^{n} \| x_i^{(j)} - c_j \|^2$$

Where, $x_j^{(j)}$ = A data point

c_j = The cluster centre

n = Number of data points

k = Number of clusters

$\| x_i^{(j)} - c_j \|^2$ = Distance measure between a data point $x_i^{(j)}$ and the cluster centre c_j

5.2.2(A) K-means Algorithm

k : number of clusters

n : sample feature vectors $x_1, x_2, ..., x_n$

m_i : the mean of the vectors in cluster i

- Assume $k < n$.
- Make initial guesses for the means $m_1, m_2, ..., m_k$.
- Until there are no changes in any mean.

Use the estimated means to classify the samples into clusters.

```
for i = 1 to k
    Replace mi with the mean of all of the samples for cluster i
    end_for
end_until
```

Following the three steps are repeated until convergence:

Iterate till no object moves to a different group :

Step 1 : Find the centroid coordinate.

Step 2 : Find the distance of each object to the centroids.

Step 3 : Based on minimum distance group the objects.

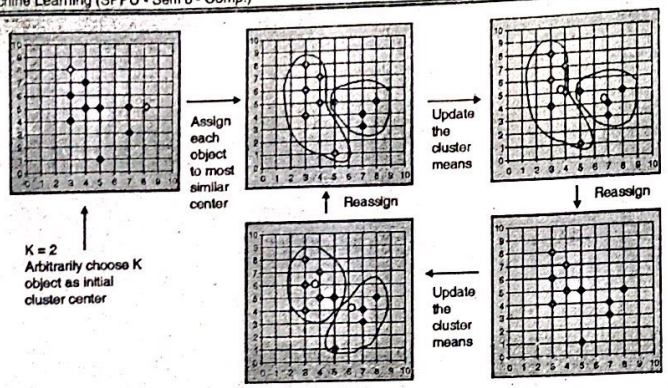


Fig. 5.2.2 : K-means graphical example

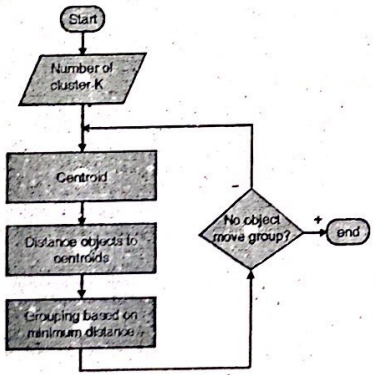


Fig. 5.2.3 : Basic steps for K-means clustering

Given a cluster $K_i = \{t_{i1}, t_{i2},..,t_{im}\}$, the cluster mean is $m_i = (1/m)(t_{i1} + ... + t_{im})$

5.2.2(B) Clustering with Scikit-learn

Let's consider a simple example with a dummy dataset. An example taken from

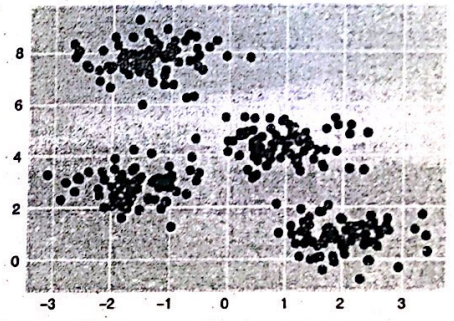
https://jakevdp.github.io/PythonDataScienceHandbook/05.11-k-means.html.

We begin with the standard imports :

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns; sns.set() # for plot styling
import numpy as np
```


First, let's generate a two-dimensional dataset containing four distinct blobs. To emphasize that this is an unsupervised algorithm, we will leave the labels out of the visualization.

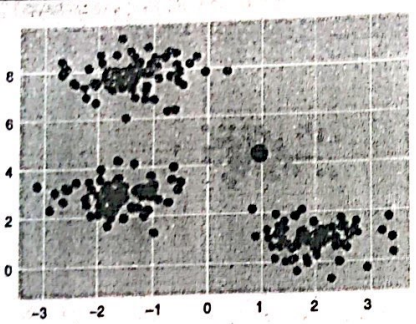
```
From sklearn.datasets.samples_generatorimport make_blobs
X, y_true = make_blobs(n_samples=300, centers=4,
                       cluster_std=0.60, random_state=0)
plt.scatter(X[:, 0], X[:, 1], s=50);
```




```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=4)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)
print(kmeans.cluster_centers_)

[[-1.37324398  7.75368871]
 [-1.58438467  2.83081263]
 [ 1.98258281  0.86771314]
 [ 0.94973532  4.41906906]]
```

Let's visualize the results by plotting the data colored by these labels

```
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')

centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5);
```



Strength of K-means clustering

Relatively efficient : O(ikn),

where n is number of objects,

k is number of clusters,

i is number of iterations.

Normally, k, i<<n.

- K-means often terminates at a **local optimum.**
- Techniques like deterministic annealing and genetic algorithms are used to get the global optimum solution.

Weakness of K-means clustering

- Applicable only when **mean** is defined.
- Need to specify k, the number of clusters, in advance.
- Unable to handle noisy data and **outliers** (outlier: objects with extremely large values)
- Not suitable to discover clusters with **non-convex shapes.**

Syllabus Topic : Finding Optimal Number of Clusters

5.2.2(C) Finding Optimal Number of Clusters

The disadvantage of k-means is to find the optimal number of clusters. If the number of clusters are less , then large element grouping can be formed with heterogeneous elements. With the more number of clusters , it will be difficult to get the dissimilarities among clusters.

Therefore, different methods to determine the number of clusters are

1. Optimizing the inertia
2. Silhouette score
3. Calinski-Harabasz index
4. Cluster instability

1. Optimizing the inertia

- The assumption that an appropriate number of clusters must produce a small inertia.
- When number of clusters is same as number of data elements, this value reaches its minimum (0.0).
- So don't look for the minimum, but for a value which is a trade-off between the inertia and the number of clusters.
- This method calculates the inertias for different number of clusters.

2. Silhouette score

- **Silhouette score is** a way to measure how the elements within a cluster are close to the points in its neighbouring clusters.
- It is based on the principal of "maximum internal cohesion and maximum cluster separation" means how similarly an object is in its own cluster (cohesion) compared to other clusters (separation).
- It finds the optimal value of k (number of clusters) during clustering.
- Define a distance metric and compute the average intra-cluster distance for each element :

$$a(\bar{x}_i) = E_{\bar{x}_j \neq C}[d(\bar{x}_i, \bar{x}_j)] \; \forall \; \bar{x}_i \in C$$

- Calculate the average nearest-cluster distance which is the lowest inter-cluster distance.

$$b(\bar{x}_i) = E_{\bar{x}_j \in D}[d(\bar{x}_i, \bar{x}_j)] \; \forall \; \bar{x}_i \in C \text{ where } D = \text{argmin}\{d(C, D)\}$$

- The silhouette score for an element x is defined as :

$$S(\bar{x}_i) = \frac{b(\bar{x}_i) - a(\bar{x}_i)}{\max\{a(\bar{x}_i), b(\bar{x}_i)\}}$$

- This value of silhouette score is bounded between -1 and 1.
- Score closer to 1 means assigned to the cluster correctly and score closer to – 1 is assigned to a wrong cluster. A score close to 0 means the point lies between almost at the boundary of both the clusters.
- scikit-learn allows computing the average silhouette score to have an immediate overview for different numbers of clusters : The program is taken from

https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html

```
from __future__ import print_function

from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score

import matplotlib.pyplot as plt
import matplotlib.cm as cm
import numpy as np

print(__doc__)

# Generating the sample data from make_blobs
```

```
# This particular setting has one distinct cluster and 3 clusters placed close
# together.
X, y = make_blobs(n_samples=500,
                  n_features=2,
                  centers=4,
                  cluster_std=1,
                  center_box=(-10.0, 10.0),
                  shuffle=True,
                  random_state=1)  # For reproducibility

range_n_clusters = [2, 3, 4, 5, 6]

for n_clusters in range_n_clusters:
    # Create a subplot with 1 row and 2 columns
    fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(18, 7)

    # The 1st subplot is the silhouette plot
    # The silhouette coefficient can range from -1, 1 but in this example all
    # lie within [-0.1, 1]
    ax1.set_xlim([-0.1, 1])
    # The (n_clusters+1)*10 is for inserting blank space between silhouette
    # plots of individual clusters, to demarcate them clearly.
    ax1.set_ylim([0, len(X) + (n_clusters + 1) * 10])

    # Initialize the clusterer with n_clusters value and a random generator
    # seed of 10 for reproducibility.
    clusterer = KMeans(n_clusters=n_clusters, random_state=10)
    cluster_labels = clusterer.fit_predict(X)

    # The silhouette_score gives the average value for all the samples.
    # This gives a perspective into the density and separation of the formed
    # clusters
    silhouette_avg = silhouette_score(X, cluster_labels)
    print("For n_clusters =", n_clusters,
          "The average silhouette_score is :", silhouette_avg)

    # Compute the silhouette scores for each sample
    sample_silhouette_values = silhouette_samples(X, cluster_labels)

    y_lower = 10
    for i in range(n_clusters):
        # Aggregate the silhouette scores for samples belonging to
```

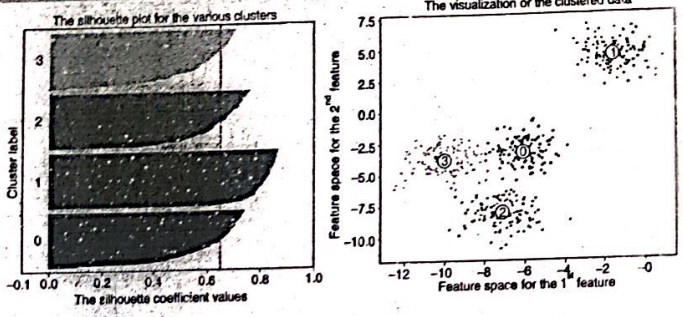


```
        # cluster i, and sort them
        ith_cluster_silhouette_values = \
            sample_silhouette_values[cluster_labels == i]

        ith_cluster_silhouette_values.sort()

        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        color = cm.nipy_spectral(float(i) / n_clusters)
        ax1.fill_betweenx(np.arange(y_lower, y_upper),
                          ith_cluster_silhouette_values,
                          facecolor=color, edgecolor=color, alpha=0.7)

        # Label the silhouette plots with their cluster numbers at the middle
        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

        # Compute the new y_lower for next plot
        y_lower = y_upper + 10  # 10 for the 0 samples

    ax1.set_title("The silhouette plot for the various clusters.")
    ax1.set_xlabel("The silhouette coefficient values")
    ax1.set_ylabel("Cluster label")

    # The vertical line for average silhouette score of all the values
    ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

    ax1.set_yticks([])  # Clear the yaxis labels / ticks
    ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])

    # 2nd Plot showing the actual clusters formed
    colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
    ax2.scatter(X[:, 0], X[:, 1], marker='.', s=30, lw=0, alpha=0.7,
                c=colors, edgecolor='k')

    # Labeling the clusters
    centers = clusterer.cluster_centers_
    # Draw white circles at cluster centers
    ax2.scatter(centers[:, 0], centers[:, 1], marker='o',
                c="white", alpha=1, s=200, edgecolor='k')

    for i, c in enumerate(centers):
        ax2.scatter(c[0], c[1], marker='$%d$' % i, alpha=1,
```

```
ax2.set_title("The visualization of the clustered data.")
ax2.set_xlabel("Feature space for the 1st feature")
ax2.set_ylabel("Feature space for the 2nd feature")

plt.suptitle("Silhouette analysis for KMeans clustering on sample data "
             "with n_clusters = %d" % n_clusters,
             fontsize=14, fontweight='bold')

plt.show()
```

Output

```
For n_clusters = 2 The average silhouette_score is : 0.7049787496083261
For n_clusters = 3 The average silhouette_score is : 0.5882004012129721
For n_clusters = 4 The average silhouette_score is : 0.6505186632729437
For n_clusters = 5 The average silhouette_score is : 0.5637646026194
For n_clusters = 6 The average silhouette_score is : 0.4504666294372765
```



Similar graphs are generated for 2, 3, 5, 6 number of clusters.

3. Calinski-Harabasz Index

- It is a concept of dense and well-separated clusters.
- It is required to define the inter cluster dispersion initially.
- The inter-cluster dispersion (BCD) is defined as :

$$BCD(k) = Tr(B_k) \quad \text{where } B_k = \sum_t n_t (\mu - \mu_t)^T (\mu - \mu_t)$$

- Where k is the cluster with their relative centroids μ_t and the global centroid μ , n_t is the number of elements belonging to the cluster k.

- The intra-cluster dispersion (WCD) is defined as :

$$WCD(k) = Tr(X_k) \quad \text{where } X_k = \sum_t \sum_{x \in C_k} (x - \mu_t)^T (x - \mu_t)$$

- The Calinski-Harabasz index is defined as the ratio between *BCD(k)* and *WCD(k)* :

$$CH(k) = \frac{N-k}{k-1} \cdot \frac{BCD(k)}{WCD(k)}$$

- We have to find the number of clusters which gives a maximum index Calinski-Harabasz index
- The code to find Calinski-Harabasz index is given below for iris dataset. Change the number of clusters in code and check the index.

```
# Refer : https://scikit-learn.org/stable/modules/clustering.html

from sklearn import metrics
from sklearn.metrics import pairwise_distances
from sklearn import datasets
dataset = datasets.load_iris()
X = dataset.data
y = dataset.target

# the Calinski-Harabaz index is applied to the results of a cluster analysis

import numpy as np
from sklearn.cluster import KMeans
# Number of clusters = 3
kmeans_model = KMeans(n_clusters=3, random_state=1).fit(X)
labels = kmeans_model.labels_
metrics.calinski_harabaz_score(X, labels)
```

Output

```
560.3999242466402
```

```
# Number of clusters = 5
kmeans_model = KMeans(n_clusters=5, random_state=1).fit(X)
labels = kmeans_model.labels_
metrics.calinski_harabaz_score(X, labels)
```

Output

```
494.0943819140987
```

4. Cluster Instability

- It is based on the concept of cluster instability defined in Von Luxburg U.
- If we have a dataset X, we can define a set of m perturbed (or noisy) versions :

$$X_n' = \{X_n^0, X_n^1, \ldots, X_n^m\}$$

- Considering a distance metric $d(C(X_i), C(X_j))$ between two clusterings with the same number (k) of clusters, the instability is defined as the average distance between couples of clusterings of noisy versions :

$$I(C) = E_{X_i, X_j \in X_n} [d(C(X_i), C(X_j))]$$

- We need to find the value of k that minimizes $I(C)$ (and therefore maximizes the stability).

Syllabus Topic : DBSCAN

5.2.3 DBSCAN

DBSCAN : Density Based Spatial Clustering of Applications with Noise

The algorithm DBSCAN, based on the formal notion of density-reachability for k-dimensional points, is designed to discover clusters of arbitrary shape. The runtime of the algorithm is of the order $O(n \log n)$ if region queries are efficiently supported by spatial index structures, i.e. at least in moderately dimensional spaces.

Explanation of DBSCAN Steps

- Epsilon (Eps) and Minimum points (MinPts) are the two parameters needed by DBSCAN. An unvisited point is chosen as the starting point. Eps between the starting point and its neighbors is calculated and the points within it are considered.
- If the number of data points in the neighbourhood is greater than or equal to MinPts then a cluster is formed. The starting point chosen is marked as visited.
- The above steps are then repeated for all the remaining neighbours.
- If the data points found in the neighbourhood are less than MinPts then they are marked as noise.
- If all the points within reach in a cluster are visited then the algorithm proceeds by choosing other remaining unvisited points in the dataset.

Basic concept

For any cluster, we have :

- A central point (p) i.e. core point.
- A distance from the core point (Eps).
- Minimum number of points within the specified distance (MinPts).

Major features

(i) Discover clusters of arbitrary shape.

(ii) Handles noise.

(iii) One scan.

(iv) Need density parameters as termination condition.

DBSCAN method

- Clusters of arbitrary shape and size are grown which are dense.
- The algorithm is as follows :

  (1) Core objects and density reachable objects are found out, merge these density reachable core objects and their clusters are discovered.

  (2) When no new points can be added to any of the clusters the execution may be stopped.

- Clusters are dense regions of objects separated by regions of low density (noise).
- Outliers will not affect the creation of cluster.

Input

(1) MinPts : Minimum number of points in any cluster.

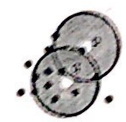
(2) $\in$: For each point in the cluster there must be another point in its less than this distance away.

- $\in$-neighborhood : Points within $\in$ distance of a point.
- $N_\in(p) : \{q$ belongs to $D \mid dist(p,q) <= \in\}$
- Core point : $\in$-neighborhood dense enough (MinPts)
- Directly density-reachable : A point p is directly density-reachable from a point q if the distance is small ($\in$) and q is a core point.

  (1) p belongs to $N_\in(q)$

  (2) core point condition :

  $|N_\in(q)| >= MinPts$



MinPts = 5

Eps = 1cm

Issues

- One of the limitations is that, the user has to set the MinPts and Eps threshold. This needs a good knowledge of the dataset. Sometimes in high dimensional data set, it is difficult to decide.
- Some of the dataset distribution may be globally inconsistent. E.g. some of the area in the dataset may be too dense compared to the other areas, some of the sections may not have clusters or noise may be present.
- The process is extremely sensitive to noise which leads to very different clusters.

Code for DBSCAN (https://scikit-learn.org/stable/auto_examples/cluster/plot_dbscan.html)

```
print(__doc__)

import numpy as np

from sklearn.cluster import DBSCAN
from sklearn import metrics
from sklearn.datasets.samples_generator import make_blobs
from sklearn.preprocessing import StandardScaler

##################################################### Generate sample data
centers = [[1, 1], [-1, -1], ...
```

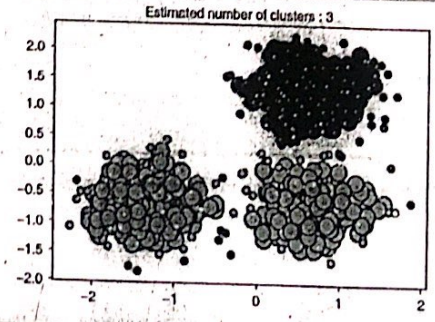
```
X,labels_true=make_blobs(n_samples=750,centers=centers,cluster_std=0.4,
random_state=0)

X=StandardScaler().fit_transform(X)

##############################################################################
# Compute DBSCAN
db=DBSCAN(eps=0.3,min_samples=10).fit(X)
core_samples_mask=np.zeros_like(db.labels_,dtype=bool)
core_samples_mask[db.core_sample_indices_]=True
labels=db.labels_

# Number of clusters in labels, ignoring noise if present.
n_clusters_=len(set(labels))-(1 if -1 in labels else 0)
n_noise_=list(labels).count(-1)

print('Estimated number of clusters: %d' % n_clusters_)
print('Estimated number of noise points: %d' % n_noise_)
print("Homogeneity: %0.3f" % metrics.homogeneity_score(labels_true,labels))
print("Completeness: %0.3f" % metrics.completeness_score(labels_true,labels))
print("V-measure: %0.3f" % metrics.v_measure_score(labels_true,labels))
print("Adjusted Rand Index: %0.3f"
%metrics.adjusted_rand_score(labels_true,labels))
print("Adjusted Mutual Information: %0.3f"
%metrics.adjusted_mutual_info_score(labels_true,labels))
print("Silhouette Coefficient: %0.3f"
%metrics.silhouette_score(X,labels))
##############################################################################
# Plot result
import matplotlib.pyplot as plt

# Black removed and is used for noise instead.
unique_labels=set(labels)
colors=[plt.cm.Spectral(each)
for each in np.linspace(0,1,len(unique_labels))]
for k,col in zip(unique_labels,colors):
if k==-1:
# Black used for noise.
col=[0,0,0,1]

class_member_mask=(labels==k)

xy=X[class_member_mask&core_samples_mask]
plt.plot(xy[:,0],xy[:,1],'o',markerfacecolor=tuple(col)
markeredgecolor='k',markersize=14)
```



```
xy=X[class_member_mask&~core_samples_mask]
plt.plot(xy[:,0],xy[:,1],'o',markerfacecolor=tuple(col),
markeredgecolor='k',markersize=6)

plt.title('Estimated number of clusters: %d'%n_clusters_)
plt.show()
```

Output

```
Automatically created module for IPython interactive environment
Estimated number of clusters: 3
Estimated number of noise points: 18
Homogeneity: 0.953
Completeness: 0.883
V-measure: 0.917
Adjusted Rand Index: 0.952
Adjusted Mutual Information: 0.883
Silhouette Coefficient: 0.626
```



Syllabus Topic : Spectral Clustering

5.2.4 Spectral Clustering

The goal of spectral clustering is to cluster data that is connected but not necessarily compact or clustered within convex boundaries.

Three main steps of spectral clustering :

1. Create a similarity graph between our *N* objects to cluster.
2. Compute the first k eigenvectors of its Laplacian matrix to define a feature vector for each object.
3. Run k-means on these features to separate objects into k classes.

- Compactness, e.g. k-means, mixture models
- Connectivity, e.g. spectral clustering

Compactness

Connectivity

Charles H Martin (https://calculatedcontent.com/2012/10/09/spectral-clustering/) has described the algorithm as given below :

1. project your data into R^n
2. define an *Affinity* matrix A, using a Gaussian Kernel K or say just an Adjacency matrix (i.e. $A_{ij} = \delta_{i,j}$).
3. construct the Graph Laplacian from A (i.e. decide on a normalization).
4. solve an Eigen value problem , such as $L_r = X_r$ (or a Generalized Eigen value problem $L_r = \lambda D_r$).
5. select k eigenvectors { v_i, i = 1, k} corresponding to the k lowest (or highest) eigenvalues { λ_i, i = 1, k}, to define a k-dimensional subspace $P^t LP$.
6. form clusters in this subspace using, say, k-means.

Syllabus Topic : Evaluation Methods Based on Ground Truth- Homogeneity, Completeness, Adjusted Rand Index

5.3 Evaluation Methods Based on Ground Truth- Homogeneity, Completeness, Adjusted Rand Index

Before the prediction of any new sample, evaluation of model is necessary. The model can be evaluated with Homogeneity, Completeness, Adjusted Rand Index.

1. Homogeneity (h)

The homogeneity score is used to determine the optimal number of clusters. A clustering result satisfies homogeneity if all of its clusters contain only data points which are members of a single class.

$$h = 1 - \frac{H(C|K)}{H(C)}$$

Where,

C : class set ,

K : Cluster set ,

H(C|K) : measure of uncertainty after clustering in determining the right class. it's necessary to normalize this value considering the initial entropy of the class set H(C).

2. Completeness (c)

Each sample of the same class must belong to the same cluster. Completeness can be measured using conditional entropy H(K|C). It's necessary to normalize this using entropy H(K). The value should be high to evaluate the number of clusters selected. If the value is high, then the majority of samples of same class assigned to the same cluster.

$$c = 1 - \frac{H(K|C)}{H(K)}$$

3. Adjusted rand index

The adjusted rand index measures the similarity between the original class partitioning and the clustering.

a : The number of pairs of elements belonging to the same partition in the class set C and to the same partition in the clustering set K number of pairs of elements belonging to different partitions in the class set C and to different partitions in the clustering set K.

If our total number of samples in the dataset are n, the rand index is defined as :

$$R = \frac{a+n}{\binom{n}{2}}$$

The Corrected for Chance version is the adjusted rand index, defined as follows :

$$AR = \frac{R - E[R]}{\max\{R] - E[R]}$$

As the adjusted rand score is bounded between -1.0 and 1.0, with negative values representing a bad situation.

- The code for Evaluation methods :

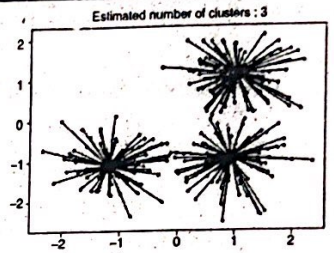
https://scikit-learn.org/stable/auto_examples/cluster/plot_affinity_propagation.html#sphx-glr-auto-examples-cluster-plot-affinity-propagation-py

```
print(__doc__)

from sklearn.cluster import AffinityPropagation
from sklearn import metrics
from sklearn.datasets.samples_generator import make_blobs

##############################################################################
# Generate sample data
centers = [[1, 1], [-1, -1], [1, -1]]
X, labels_true = make_blobs(n_samples=300, centers=centers, cluster_std=0.5,
                            random_state=0)

##############################################################################
# Compute Affinity Propagation
af = AffinityPropagation(preference=-50).fit(X)
cluster_centers_indices = af.cluster_centers_indices_
labels = af.labels_

n_clusters_ = len(cluster_centers_indices)
```

```
print('Estimated number of clusters: %d'%n_clusters_)
print("Homogeneity: %0.3f"%metrics.homogeneity_score(labels_true,la.els))
print("Completeness: %0.3f"%metrics.completeness_score(labels_true,labels))
print("V-measure: %0.3f"%metrics.v_measure_score(labels_true,labels))
print("Adjusted Rand Index: %0.3f"
%metrics.adjusted_rand_score(labels_true,labels))
print("Adjusted Mutual Information: %0.3f"
%metrics.adjusted_mutual_info_score(labels_true,labels))
print("Silhouette Coefficient: %0.3f"
%metrics.silhouette_score(X,labels,metric='sqeuclidean'))

# ############################################################# Plot result
import matplotlib.pyplot as plt
from itertools import cycle
plt.close('all')
plt.figure(1)
plt.clf()

colors = cycle('bgrcmykbgrcmykbgrcmykbgrcmyk')
for k, col in zip(range(n_clusters_), colors):
class_members = labels == k
cluster_center = X[cluster_centers_indices[k]]
plt.plot(X[class_members, 0], X[class_members, 1], col + '.')
plt.plot(cluster_center[0], cluster_center[1], 'o', markerfacecolor=col,
markeredgecolor='k', markersize=14)
for x in X[class_members]:
plt.plot([cluster_center[0], x[0]], [cluster_center[1], x[1]], col)

plt.title('Estimated number of clusters: %d' % n_clusters_)
plt.show()
```

Output

```
Automatically created module for IPython interactive environment
Estimated number of clusters: 3
Homogeneity: 0.872
Completeness: 0.872
V-measure: 0.872
Adjusted Rand Index: 0.912
Adjusted Mutual Information: 0.871
Silhouette Coefficient: 0.753
```

Syllabus Topic : Introduction to Meta Classifier

5.4 Introduction to Meta Classifier

- Meta classifier is a classifier, which is usually a proxy to the main classifier, used to provide additional data pre-processing.
- Meta classifier doesn't implement a classification algorithm on its own, but uses another classifier to do the actual work. In addition, the meta-classifier adds another processing step that is performed before the actual base-classifiersees the data.
- Use meta classifier before the classification on both training and testing data. The Meta classifier technique is used to implement arbitrary pre-processing within a cross-validation analysis.

Syllabus Topic : Concepts of Weak and Eager Learner

5.4.1 Concepts of Weak and Eager Learner

- **A weak learner :** It is defined to be a classifier that is only slightly correlated with the true classification. In contrast, a strong learner is a classifier that is arbitrarily well-correlated with the true classification.
- **Eager learner :** Given a set of training tuples, constructs a classification model before receiving new (e.g., test) data to classify.

Review Questions

Q. 1 Explain different impurity measures.

Q. 2 What is Ensemble Learning?

Q. 3 Explain following algorithms in detail

(a) Random Forest (b) Ada Boost

(c) K-Means (d) DBSCAN

Q. 4 What are different methods to determine number of clusters? Explain.

Chapter Ends...

UNIT VI

CHAPTER 6

Clustering Techniques

Syllabus Topics

Hierarchical Clustering, Expectation maximization clustering, Agglomerative Clustering - Dendrograms, Agglomerative clustering in Scikit- learn, Connectivity Constraints.

Introduction to Recommendation Systems - Naïve User based systems, Content based Systems, Model free collaborative filtering-singular value decomposition, alternating least squares.

Fundamentals of Deep Networks - Defining Deep learning, common architectural principles of deep networks, the building blocks of deep networks.

Clustering is an unsupervised learning as classes are not known. It groups the object or data elements without having advance knowledge of group definitions. These groups or partitions of data after clustering are called clusters. Clusters have high inter similarity and low intra similarity.

Syllabus Topic : Hierarchical Clustering

6.1 Hierarchical Clustering

The set of data objects is decomposed hierarchically using certain criteria. In hierarchical clustering, algorithms either follow the top down or bottom up approach.

Hierarchical clustering technique (Basic algorithm)

1. Calculate the distance and find a distance matrix or proximity matrix
2. Consider each data point or object as a cluster.
3. Repeat.
4. Combine the two closest clusters.
5. Update the proximity matrix.
6. Until only a single cluster remains.

Note : Proximity matrix is symmetric, meaning that the numbers on the lower half of the diagonal will be the same as the numbers on the top half of the diagonal.

Agglomerative Clustering

- In Bottom up approach, every object is considered to be a cluster and in subsequent iterations they are merged into a single cluster. Therefore, it is also called as Hierarchical Agglomerative Clustering (HAC).

Divisive Hierarchical Clustering

In this data objects are grouped in a top down manner. Initially, all objects are in one cluster. Then the cluster is subdivided into smaller and smaller pieces, until each object forms a cluster on its own or until it satisfies certain termination conditions as the desired number of clusters are obtained. Divisive methods are not generally available, and rarely have been applied.

Difference between Agglomerative and Divisive Hierarchical Clustering

Agglomerative	Divisive
Start by considering each individual element as a cluster.	Start with only one cluster by combining all elements.
At every step, clusters are merged based on closest pair until a single or k clusters are left	At every step, the cluster is divided or split until each cluster contains a single element or k clusters are left.

Advantages

- This algorithm is simple and gives an output as a hierarchy
- The structure obtained is easy to understand and more informative.
- There is no need to pre-specify the number of clusters

Disadvantages

- Merging and splitting is critical once the clusters are formed, as undo is not possible. Every time it performs on newly generated clusters.
- If decision to merge and split is wrong, then low quality clusters are formed
- Scikit-learn implements only the agglomerative clustering as the complexity of divisive is higher than Agglomerative and has similar performance of Divisive approach.

Syllabus Topic : Expectation Maximization Clustering

6.2 Expectation Maximization Clustering

- The Expectation Maximization (EM) algorithm can be applied to bring forth the best theory for the distributional parameters of some multi-modal data.
- The best hypothesis for the distributional parameters is the maximum likelihood hypothesis – the one that maximizes the probability that this data we are looking at comes from K distributions, each with a mean m_i and variance $sigma_i^2$
- The Expectation Maximization (EM) Algorithm deal with missing labels by alternating between two steps :
  1. **Expectation (E) :** Fix model and estimate missing labels
  2. **Maximization (M) :** Fix missing labels (or a distribution over the missing labels) and find the model that maximizes the expected log-likelihood of the data.

General EM Algorithm

- The alternate steps until model parameters don't change much :
  - **E step :** Estimate distribution over labels given a certain fixed model.
  - **M step :** Choose new parameters for model to maximize expected log-likelihood of observed data and hidden variables.

Formal Setup for General EM Algorithm

Let $D = \{x^{(1)},...x^{(n)}\}$ be n observed data vectors.

Let $Z = \{z^{(1)},...z^{(n)}\}$ be n values of hidden variables (i.e. the cluster labels)

Log-likelihood of observed data given model :

$$L(\theta) = \log p(D|\theta) = \log \sum_{Z} p(D,Z|\theta)$$

Note : both θ and Z are unknown.

Where theta is a vector of unknown parameters.

In clustering, K (number of clusters) are not known initially. We can produce two results for the clustering :

- **Hard clustering :** In this method , data object or observation i belongs to only one cluster, so clusters can be produced like { C1, C2, ...Ck}.
- **Soft clustering :** The method says that data object or observation is more likely to belong to one of the K cluster by producing a probability distribution :

$$P[X_i \in C_k] = \gamma_{k,i} \qquad \text{with} \sum_{k=1}^{K} \gamma_{k,i} = 1$$

From Fig. 6.2.1, we can observe that hard clustering is possible on the left side figure. But right side figure is not clear, so only based on probability samples can be classified to one of the class. This is the purpose of soft clustering. Algorithms like hierarchical clustering or K means produce some Hard clustering. The purpose of the EM clustering is to propose a Soft clustering method.

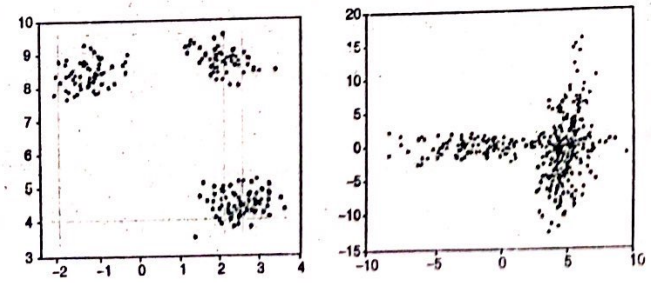


Fig. 6.2.1 : Visual point of view hard clustering and soft clustering

The basic steps for the algorithm are :

1. An initial guess is made for the model's parameters and a probability distribution is made. This is sometimes called the **"E-Step" for the "Expected" distribution".**
2. Newly observed data is fed into the model.
3. The chance distribution of the E-step is tweaked to let in the fresh information. This is sometimes called the "M-step".
4. Steps 2 through 4 are repeated until stability (i.e. a distribution that doesn't change from the E-step to the M-step) is reached.

Syllabus Topic : Agglomerative Clustering

6.3 Agglomerative Clustering

6.3.1 Affinity

Affinity is a metric function of two arguments with the same dimensionality m.

The most common metrics supported by scikit-learn are :

1. **Euclidean :** Euclid stated that the shortest distance between two points is a line and thus it is predominantly known as **Euclidean distance.** It was often called **Pythagorean metric** since it is derived from the Pythagorean Theorem.

$$d_{Euc} = \sqrt{\sum_{i=1}^{d} |P_i - Q_i|^2}$$

So the distance between two points is

$$\text{dist}((x, y), (a, b)) = \sqrt{(x-a)^2 + (y-b)^2}$$

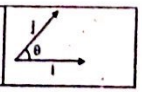
2. **Manhattan or City Block:** The City block distance between two points, C and B, with d dimensions are calculated as

$$d_{CB} = \sum_{i=1}^{d} |P_i - Q_i|$$

The distance between two points measured along axes at right angles. In a plane with p_1 at (x_1, y_1) and p_2 at (x_2, y_2), it is $|x_1 - x_2| + |y_1 - y_2|$.

3. **Cosine distance :** Cosine similarity is a measure of similarity between two vectors. The data objects are treated as vectors. Similarity is measured as the angle θ between the two vectors. Similarity is 1 when θ = 0, and 0 when θ = 90°.

Similarity function is given by

$\cos(i,j) = \frac{i \bullet j}{\|i\| \times \|j\|}$	$i \bullet j = \sum_{k-1}^{n} i_k j_k$	$\|i\| = \sqrt{\sum_{k-1}^{n} i_k^2}$	

Cosine Distance = 1 – Cosine Similarity

$$d_{cosine}(\bar{x}_1, \bar{x}_2) = 1 - \frac{\bar{x}_1 \cdot \bar{x}_2}{\|\bar{x}_1\|_2 \|\bar{x}_2\|_2}$$

- The cosine distance is useful when we need a distance proportional to the angle between two vectors. If the direction is the same, the distance is null, while it is maximum when the angle is equal to 180° (meaning opposite directions).

6.3.2 Strategy to Aggregate Different Clusters

Different approaches to defining the distance between clusters distinguish the different algorithms, but scikit learn supports the three most common ones i.e. Complete Linkage, Average Linkage and Ward's linkage.

1. **Single-linkage**

- Single Linkage clustering is also called as minimum method, the minimum distance from any object of one cluster to any object of another cluster is considered. In the single linkage method, **D(A,B)** is computed as

  D(A,B) = Min {d(i,j)

  Where object i is in cluster A and object j is in cluster B}
- This measure of inter-group distance is illustrated in the Fig. 6.3.1.

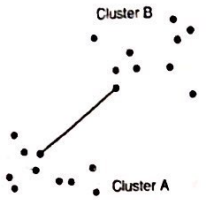


Fig. 6.3.1 : Single-linkage clustering

2. **Complete linkage**

- **Complete linkage** also called as maximum method, the maximum distance between any object of one cluster to any object of another cluster is considered.
- In the complete linkage method, **D(A,B)** is computed as

  D(A,B) = Max { d(i, j) :

  Where object i is in cluster A and object j is in cluster B}
- The measure is illustrated in the Fig. 6.3.2.

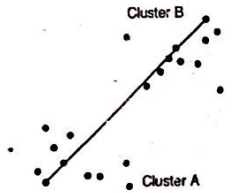


Fig. 6.3.2 : Complete-linkage clustering

3. **Average-linkage**

- In average-linkage clustering, we consider the distance between any two clusters A and B is taken to be the average of all distances between pairs of objects "i" in A and "j" in B, that is, the mean distance between elements of each cluster.
- In the average linkage method, **D(A,B)** is computed as

  D(A,B) = mean{d(i,j) : Where object i is in cluster A and object j is in cluster B}
- The Fig. 6.3.3 illustrates average linkage clustering.

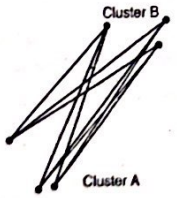
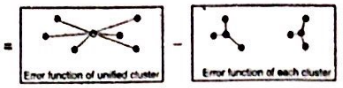


Fig. 6.3.3 : Average-linkage clustering

4. **Ward's linkage**

- In this method, the error function is defined for every cluster. This error function is the average (RMS) distance of each data-point in a cluster to the centre of gravity in the cluster.



- The distance (D) between two clusters is defined as the error function of the unified cluster minus the error functions of the individual clusters.
- The ward's linkage method computes the distance using formula,

$$\forall C_i, C_j \; L_{ij} = \sum_{x_a \in C_i} \sum_{x_b \in C_j} \|x_a - x_b\|^2$$

- The Ward's linkage supports only the Euclidean distance.

5. **Centroid clustering**

- In centroid method, the distance between two clusters is calculated by finding the distance between two centroids (i.e. mean value of cluster) of the clusters. At every step, two clusters are combined that have minimum centroid distance.
- In the centroid clustering method, D(A,B) is computed as

  $D(A,B) = d(A_{centroid}, B_{centroid})$

  Where $A_{centroid}$ is the mean value or centroid of cluster A and $B_{centroid}$ is the mean value or centroid of cluster B. The Fig. 6.3.4 illustrates centroid clustering.

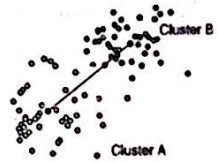


Fig. 6.3.4 : Centroid clustering

Syllabus Topic : Dendrograms

6.3.3 Dendrograms

- An agglomerative clustering is typically visualized as a *dendrogram* as shown in Fig. 6.3.5 where each merge is represented by a horizontal line.
- Dendrogram is the aggregation of clusters from bottom to top.
- Initially leaf nodes are single clusters, which are merged till a single root node or cluster generated.

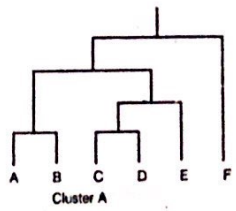


Fig. 6.3.5 : Dendrogram

What is Dendrogram?

- **Dendrogram :** A tree data structure which illustrates hierarchical clustering techniques.
- Each level shows clusters for that level.
  - Leaf – individual clusters
  - Root – one cluster
- A cluster at level i is the union of its children clusters at level i + 1.

- A clustering of the data objects is obtained by cutting the dendrogram at the desired level, then each connected component form a cluster.
- Unfortunately, scikit-learn doesn't support the dendrogram but SciPy provides some useful built-in functions.
- The program in Python is given below to create dendrogram.

```
# Program to visualize dendrogram
from sklearn.datasets import make_blobs
nb_samples = 25
X, Y = make_blobs(n_samples=nb_samples, n_features=2, centers=3,cluster_std=1.5)
# computing a distance matrix chosen a Euclidean metric
from scipy.spatial.distance import pdist
Xdist = pdist(X, metric='euclidean')
# linkage used is Ward

from scipy.cluster.hierarchy import linkage
XI = linkage(Xdist, method='ward')
# create and visualize a dendrogram
from scipy.cluster.hierarchy import dendrogram
Xd = dendrogram(XI)
```

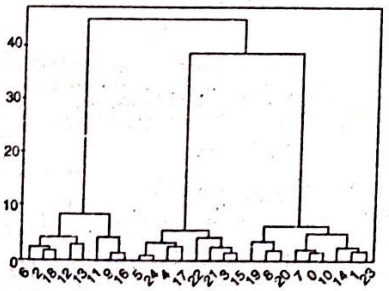


Fig. 6.3.6 : Snapshot of resulting plot of dendrogram

Syllabus Topic : Agglomerative Clustering in Scikit - learn

6.3.4 Agglomerative Clustering in Scikit-learn

Consider a more complex dummy dataset with 4 centers.

```
from sklearn.datasets import make_blobs
from sklearn.cluster import AgglomerativeClustering
import matplotlib.pyplot as plt
# We have now a data set, with 3000 randomized samples around the 4 center points with a standard deviation of 2 for now
```



```
nb_samples = 3000
X, _ = make_blobs(n_samples=nb_samples, n_features=2, centers=4,cluster_std=2.0)
# visualize the data set
plt.scatter(X[:,0],X[:,1])
plt.show()
```

A graphical representation is shown in the following Fig. 6.3.7.

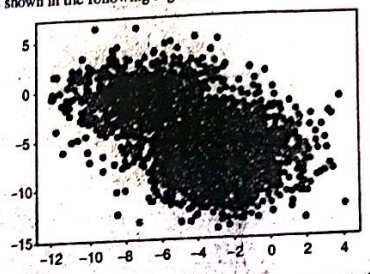


Fig. 6.3.7 : snapshot of resulting plot of data representation

X is the dataset, and y is the label of each data point, according to the sample generation. Agglomerative Clustering uses the method fit_predict() to train the model and transform the original dataset. The number of parameters is set to 4 using the n_clusters parameter while the affinity is set to "Euclidean" (distance between the data points). Finally linkage parameter is set to "complete ".

```
ac = AgglomerativeClustering(n_clusters=4, affinity='euclidean', linkage='complete')
Y = ac.fit_predict(X)
plt.scatter(X[:,0],X[:,1], c=ac.labels_, cmap='rainbow')
```

A graphical representation is shown in the following Fig. 6.3.8.

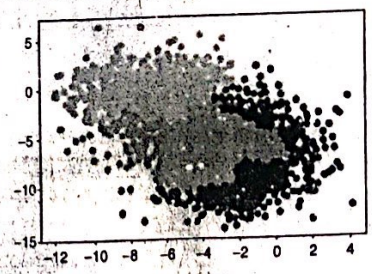


Fig. 6.3.8 : Snapshot of resulting plot of complete linkage

Linkage parameter is set to "average".

```
ac = AgglomerativeClustering(n_clusters=4, affinity='euclidean', linkage='average')
Y = ac.fit_predict(X)
plt.scatter(X[:,0],X[:,1], c=ac.labels_, cmap='rainbow')
```

The resultant plot is shown in Fig. 6.3.9 :

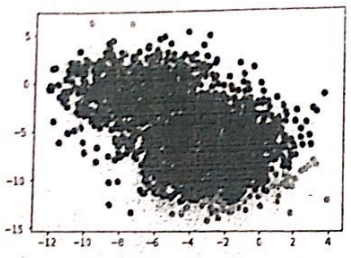


Fig. 6.3.9 : Snapshot of resulting plot of average linkage

- The last method, which is often the best (it's the default one), is Ward's linkage, that can be used only with a Euclidean metric (also the default one) :

```
ac = AgglomerativeClustering(n_clusters=20, linkage='ward')
ac.fit(X)
plt.scatter(X[:,0],X[:,1], c=ac.labels_, cmap='rainbow')
```

The resultant plot is given in Fig. 6.3.10.

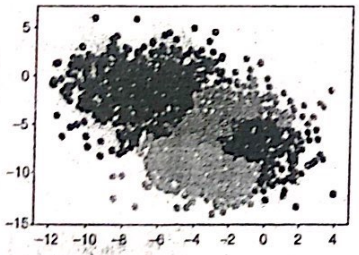


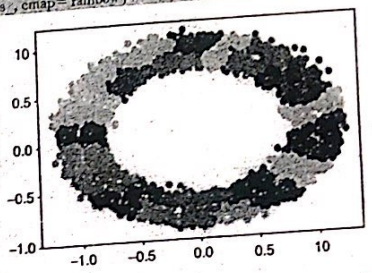
Fig. 6.3.10 : Snapshot of resulting plot of ward linkage

Syllabus Topic : Connectivity Constraints

6.3.5 Connectivity Constraints

- An interesting aspect of Agglomerative Clustering is that connectivity constraints which can be added to this algorithm where only adjacent clusters can be merged together and those clusters which are distant i.e. non-adjacent are skipped. Connectivity matrix can be used as a constraint by Scikit-learn and find the clusters to merge. These constraints are useful to impose a certain local structure, but they also make the algorithm faster, especially when the number of the samples is high.
- K-Nearest Neighbour graph function is a common method used.
- Consider circular dummy dataset in the example given below.


```
from sklearn.datasets import make_circles
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering
nb_samples = 3000
X, _ = make_circles(n_samples=nb_samples, noise=0.05)

ac = AgglomerativeClustering(n_clusters=20, linkage='average')
ac.fit(X)
plt.scatter(X[:,0],X[:,1], c=ac.labels_, cmap='rainbow')
```



```
#Now we can try to impose a constraint with different values for k:
from sklearn.neighbors import kneighbors_graph
ace = []
k = [50, 100, 200, 500]
for i in range(4):
    kng = kneighbors_graph(X, k[i])
    ac1 = AgglomerativeClustering(n_clusters=20, connectivity=kng,linkage='average')
    ac1.fit(X)
    plt.scatter(X[:,0],X[:,1], c=ac1.labels_, cmap='rainbow')
    ace.append(ac1)
```

- Imposing a constraint based on K-Nearest Neighbours, allows controlling how the agglomeration creates new clust and can be a powerful tool for tuning the models, or for avoiding elements whose distance is large in the original sp could be taken into account during the merging phase. It is helpful in clustering the images.

Syllabus Topic : Introduction to Recommendation Systems - Naïve User based Systems

6.4 Introduction to Recommendation Systems

- Recommendation system is used to predict future preferences of a set of items and also recommend the top n item users.
- Many e-commerce websites uses recommendation system to increase their sales by using the power of data.
- This system finds the user's interest to recommend items.

- Traditionally, there are two methods to construct a recommendation system :
  1. User or Content-based recommendation
  2. Collaborative Filtering

1. **Content-based recommendation system:** It makes recommendations using a user's item and profile features. To find the user's future interest, the system needs the information about user's past interest in the items. So only similar items can be grouped together based on their features.
2. **Collaborative Filtering systems:** It filters the items in which we are interested. This system is based on the assumption that if user1 likes item X and user2 likes the same item X plus another item Y, then user1 may be interested in item Y. So system can use historical data to predict new interactions.

6.4.1 Naïve User based Systems

- Let us consider a set of users represented by feature vectors :

$$U = \{\bar{u}_1, \bar{u}_2, ..., \bar{u}_n\} \text{ where } \bar{u}_n \in \mathbb{R}^n$$

- Assume the features are age, gender, interests, and so on. Set of item I is :

$$I = \{i_1, i_2, ..., i_m\}$$

- Every user is associated with a subset of items or items can be viewed or feedback has been taken for those items. So there is a relationship between user and items.

$$g(\bar{u}) \rightarrow \{i_1, i_2, ..., i_k\} \quad \text{where } k \in (0, m)$$

- In a user-based system, the users are periodically clustered (normally using a **K-Nearest Neighbour** approach), and therefore, considering a generic user **u** (also new), we can immediately determine all the users who are similar (therefore neighbours) to our sample.

$$B_R(\bar{u}) = \{\bar{u}_1, \bar{u}_2, ..., \bar{u}_k\}$$

- At this point, we can create the set of suggested items using the relation previously introduced :

$$I_{suggested}(u) = \left\{ \bigcup_j g(\bar{u}_j) \text{ where } \bar{u}_j \in B_R \bar{u} \right\}$$

- This set contains all the unique products positively rated or bought by the neighbourhood.

Syllabus Topic : Content Based Systems

6.4.2 Content Based Systems

- Recommendations are based on information on the content of items rather than on other users' opinions.
- Uses machine learning algorithms to induce a profile of the user's preferences from examples based on content features.
  - No need for data about other users.
  - No cold-start or sparsity problems.
  - Able to recommend to users with unique tastes.
  - No first-rater problem.
- **Cold Start :** enough users in the system to find a match.
- **Sparsity :** The user/ratings matrix is sparse, and it is hard to find users that have rated the same items.

- **First Rater :** Not for an item that has not been previously rated.
- This is one of the simplest methods which is based on products and modelled as feature vectors :

$$I = \{\bar{I}_1, \bar{I}_2, ..., \bar{I}_n\} \text{ where } \bar{I}_n \in \mathbb{R}^n$$

- The features can also be categorized like users. For example, the type of a book or a movie, and they can be used together with numerical values (like price, length, number of positive reviews, and so on) after encoding them.

Syllabus Topic : Model Free Collaborative Filtering

6.4.3 Model Free Collaborative Filtering

- It is based on rating among the users and "lightweight" filtering approach. It is computed in-memory without the use of complex algorithms like ALS (Alternating Least Squares) which executes in parallel environment.
- Assume that we have M products and N users, then user-preference sparse matrix is given below.

$$U_{pref} = \begin{pmatrix} u_0^{(0)} & \cdots & u_0^{(M)} \\ \vdots & \ddots & \vdots \\ u_N^{(0)} & \cdots & u_N^{(M)} \end{pmatrix}$$

- Each element from the above represents the rating given by the user i to the item j.
- 0 means there is no rating
- Two kinds of ratings or feedbacks
  1. **Explicit feedback :** An actual rating (like 3 stars or 8 out of 10)
  2. **Implicit feedback :** A page view, song play, move play and so forth
- After the user preference matrix, compute the affinity matrix based on a similarity metric using Euclidean as given below :

$$A = \begin{pmatrix} d(u_0, u_0) & \cdots & d(u_N, u_0) \\ \vdots & \ddots & \vdots \\ d(u_0, u_N) & \cdots & d(u_N, u_N) \end{pmatrix}$$

- For explicit feedback, the Euclidean metric is preferable, but when it's implicit (like page views, song/movie plays, etc.), it has a different meaning and need another metric function. For example, if we have a binary encoding like 0 and 1 (0 means no interaction and 1 means single or multiple interaction), then Hamming distance is more appropriate. Our goal is to cluster the users according to their rating. After clustering, we can check which products have higher rating for a given user and therefore are more likely to be bought.
- Once the top neighbors determined, union of most rated neighbour products is used to suggest the list of top neighbours :

$$P_{sugg}(i) = \left\{ \bigcup_{j \neq i} P(j) \right\}$$

The code to create a distance matrix using sklearn for 10 users and 10 items.

```
from scipy.sparse import dok_matrix
from sklearn.metrics.pairwise import pairwise_distances

import numpy as np

# Set random seed (for reproducibility)
np.random.seed(50)

# Create a dummy user-item dataset
nb_users = 10
nb_products = 10
max_rating = 5
max_rated_products = 8

X_preferences = dok_matrix((nb_users, nb_products), dtype=np.uint8)

for i in range(nb_users):
    # Extract n random products
    n_products = np.random.randint(0, max_rated_products + 1)
    products = np.random.randint(0, nb_products, size=n_products)
    # Populate preference sparse matrix
    for p in products:
        X_preferences[i, p] = np.random.randint(0, max_rating + 1)

# Compute pairwise distances
distance_matrix = pairwise_distances(X_preferences, metric='euclidean')
# Sort distances
sorted_distances = np.argsort(distance_matrix, axis=1)
print(sorted_distances)
```

```
[[0 1 5 7 4 8 6 2 9 3]
 [0 1 3 7 4 8 6 2 9 5]
 [2 0 1 5 7 4 8 3 6 9]
 [3 2 4 0 1 5 7 9 8 6]
 [4 0 1 5 7 8 6 2 9 3]
 [0 1 5 7 4 8 6 2 9 3]
 [6 0 1 5 7 4 8 9 2 3]
 [0 1 5 7 4 8 6 2 9 3]
 [8 0 1 5 7 9 4 6 2 3]
 [9 8 0 1 6 6 7 4 2 3]]
```

```
test_user=5
```



```
# Take the top-5 similar users
for d in sorted_distances[test_user][::-1][0:5]:
    print(d)
```

Output

```
```

Syllabus Topic : Singular Value Decomposition

6.4.4 Singular Value Decomposition

- Singular value decomposition (SVD) is a method of decomposing a matrix into three other matrices.
- Given any $m \times n$ matrix **A**, algorithm to find matrices **U**, **V**, and **W** such that

$$A = U W V^T$$

U is $m \times n$ and orthonormal

W is $n \times n$ and diagonal

V is $n \times n$ and orthonormal

$$(A) = (U) \begin{pmatrix} W_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & W_n \end{pmatrix} (V)^T$$

- By assuming m ≥ n rewrite above equation using summation notation

$$a_{ij} = \sum_{k=1}^{n} u_{ik} s_k v_{jk}$$

- Simplify the expression into a single summation. The variables, $\{s_i\}$, are called singular values and are normally arranged from largest to smallest.

$$S_{i+1} \leq S_i$$

- The columns of U are called left singular vectors, while those of V are called **right singular vectors.**
- We know that U and V are orthogonal, that is:

$$U^T U = V V^T = I$$

Where I is the identity matrix (the diagonals of the identity matrix are 1, other values being 0).

- svd() function calculate SVD.
- svd() function takes matrix as input and returns U, Sigma and V^T

- The example below defines a 3 × 2 matrix and calculates the Singular-value decomposition.

```
# Singular-value decomposition
from numpy import array
from scipy.linalg import svd
# define a matrix
A = array([[1, 2], [3, 4], [5, 6]])
print(A)
# SVD
U, s, VT = svd(A)
print(U)
print(s)
print(V)
```

Output

```
A matrix:
[[1 2]
 [3 4]
 [5 6]]
U matrix:
[[ 0.2298477   0.88346102  0.40824829]
 [ 0.52474482  0.24078249 -0.81649658]
 [ 0.81964194 -0.40189603  0.40824829]]
s matrix:
[9.5255181  0.51430058]
V matrix:
[[ 0.61962948 -0.78489445]
 [ 0.78489445  0.61962948]]
```

Syllabus Topic : Alternating Least Squares

6.4.5 Alternating Least Squares

- Alternating Least Squares (ALS) is a model use to fit our data and find similarities.
- Least square optimization problem is defined as a loss function to find the latent factor as given below.

$$L = \sum_{(i,j)} \left(r_{ij} - \bar{p}_i \cdot \bar{q}_j^T\right)^2 + \alpha\left(\|\bar{p}_i\|^2 + \|\bar{q}_j\|^2\right)$$

- L can be identified by using known samples (user, item).
- The second term of the above equation is a regularization factor.
- Any optimization method can be used to solve the whole problem.
- Two main iterating steps of algorithm are :
  1. p_i is fixed and q_j is optimized
  2. q_j is fixed and p_i is optimized

Syllabus Topic : Fundamentals of Deep Networks - Defining Deep Learning

6.5 Fundamentals of Deep Networks

- A machine learning subfield of learning representations of data: Exceptionally effective at learning patterns.
- Deep learning algorithms attempt to learn (multiple levels of) representation by using a hierarchy of multiple layers.
- If you provide the system, tons of information, it begins to understand it and respond in useful ways.

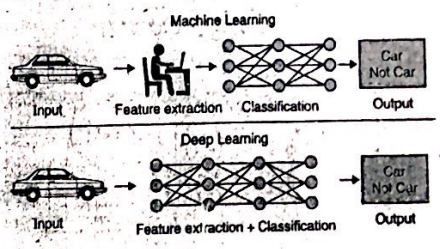


Fig. 6.5.1

6.5.1 Defining Deep Learning

'Deep Learning' means using a neural network with several layers of nodes between input and output. It is the series of layers between input and output for feature identification and processing in a series of stages, just as our brains seem to.

Uses of Deep Learning

- Annually designed features are incomplete and takes long time, but learned features are adaptable and fast to learn.
- Deep learning provides a very flexible, (almost?) Universal, a learnable framework for representing the world, visual and linguistic information. Can learn both unsupervised and supervised.
- Effective end-to-end joint system learning utilize large amounts of training data.

Syllabus Topic : Common Architectural Principles of Deep Networks

6.5.2 Common Architectural Principles of Deep Networks

How do We Train Deep Architectures?

- It's an inspiration from mammal brain.
- Deep network has Multiple Layers of "neurons" (Rumelhart et al 1986).
- Training for each layer is required to learn higher level abstraction.

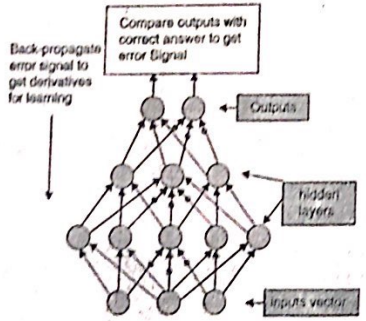
  Ex : Pixels → Edges → Contours → Object parts → Object categories → Local Features → Global Features

- Greedy strategy is also used to train the layers one-by-one (Hinton et al 2006)

6.5.2(A) Multilayer Perceptron with Back-propagation : First Deep Learning Model

(Rumelhart, Hinton, Williams 1986)

Multilayer Perceptrons with Back-propagation
First deep learning model (Rumelhart, Hinton, Williams 1986)



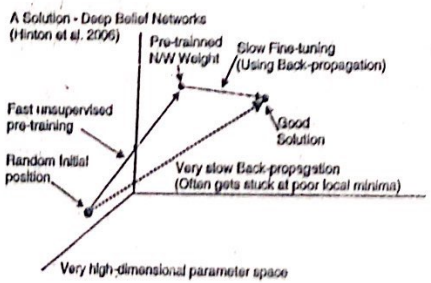
Drawbacks of Back-propagation based Deep Neural Networks

- They are discriminative models.
- All the information is collected from labels.
- But the labels are not able to give all information.
- It requires a large amount of labeled data.
- Gradient descent with random initialization leads to poor local minima.

6.5.2(B) Deep Belief Networks (Hinton et al. 2006)

- Pre-training of network is required before using back-propagation.
- Use the weights of the pre-trained network as the initial point for the traditional back-propagation.
  - This leads to quicker convergence to a better solution.

Pre-training is fast; fine-tuning can be slow.



6.5.2(C) Five Popular and Widely-used Deep Learning Architectures

1. Convolutional Neural Networks
2. Recurrent Neural Networks
3. Autoencoders
4. Generative Adversarial Networks
5. ResNets

Syllabus Topic : Building Blocks of Deep Networks

6.5.2(D) Building Blocks of Deep Networks

- Activation functions : linear, ReLU, sigmoid, tanh, etc.
- Layers : Fully connected, convolutional and pooling, recurrent (Problem: The graph is not acyclic. To compute the gradients, we unroll the computational cycle over timesteps, and backpropagate through this structure.), resnet (skip connections over layers), etc.

Review Questions

Q.1 Explain Hierarchical Clustering basic algorithm.

Q.2 Explain Expectation Maximization Clustering algorithms in detail.

Q.3 What are various strategy to Aggregate Different Clusters.

Q.4 Explain Model Free Collaborative Filtering.

Q.5 Write short note on Recommendation Systems

Q.6 What is Deep Learning? Explain.

Chapter Ends...

□□□